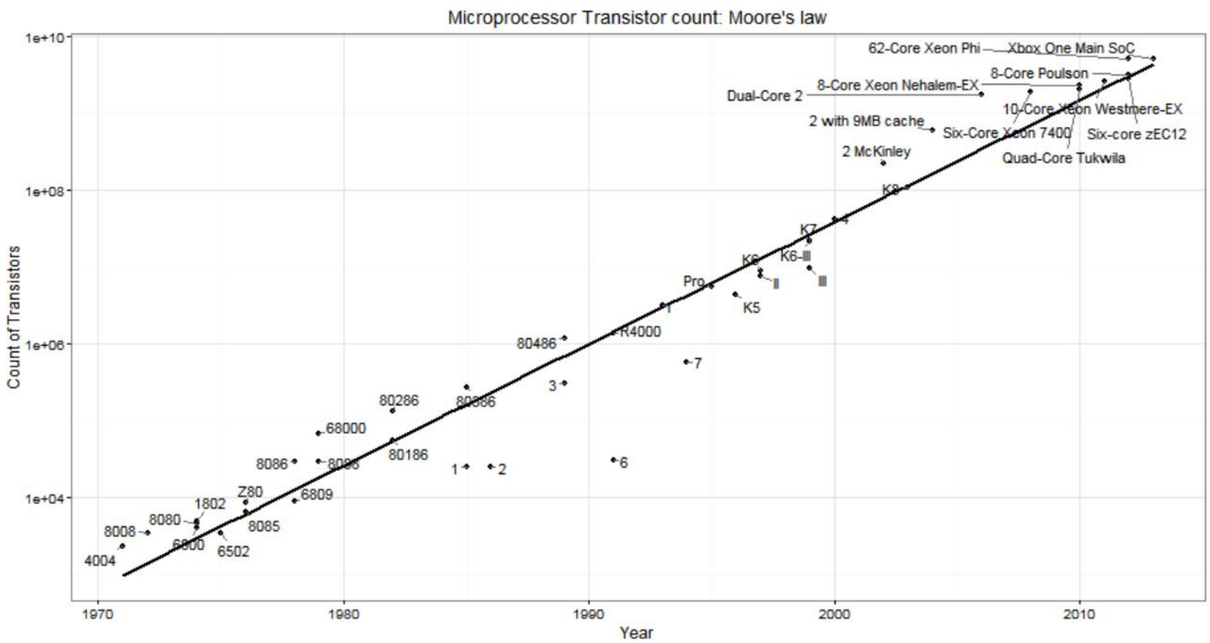
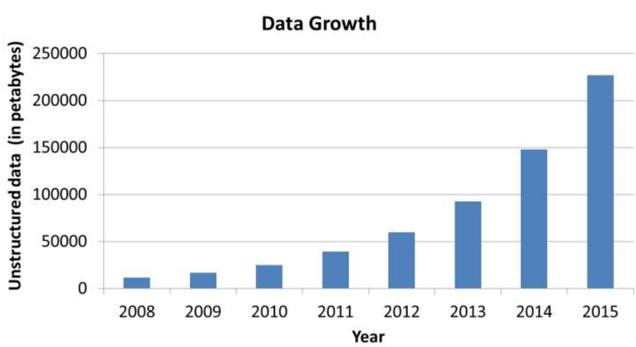
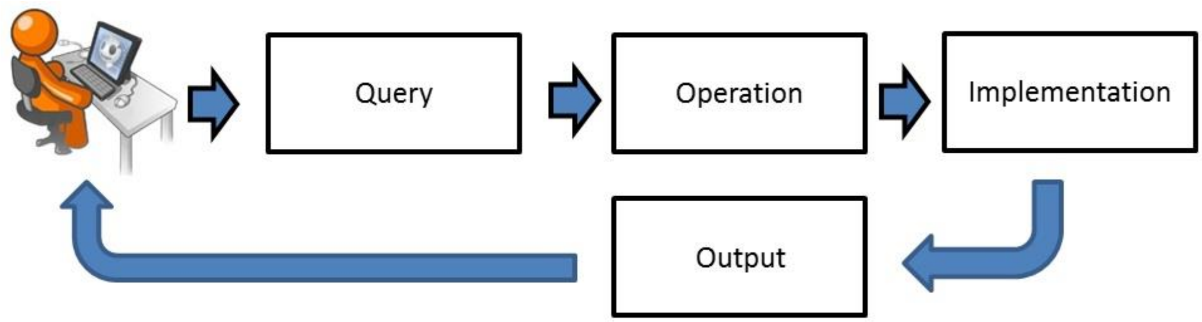
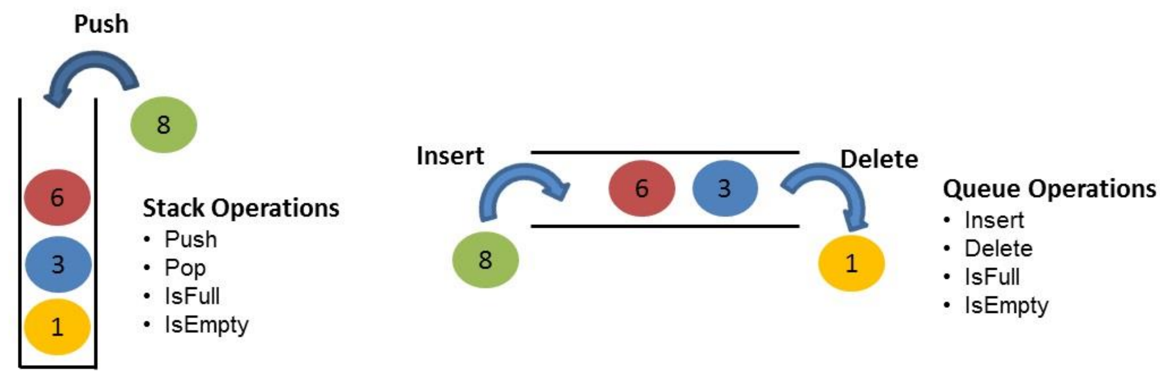
Chapter 1: Getting Started



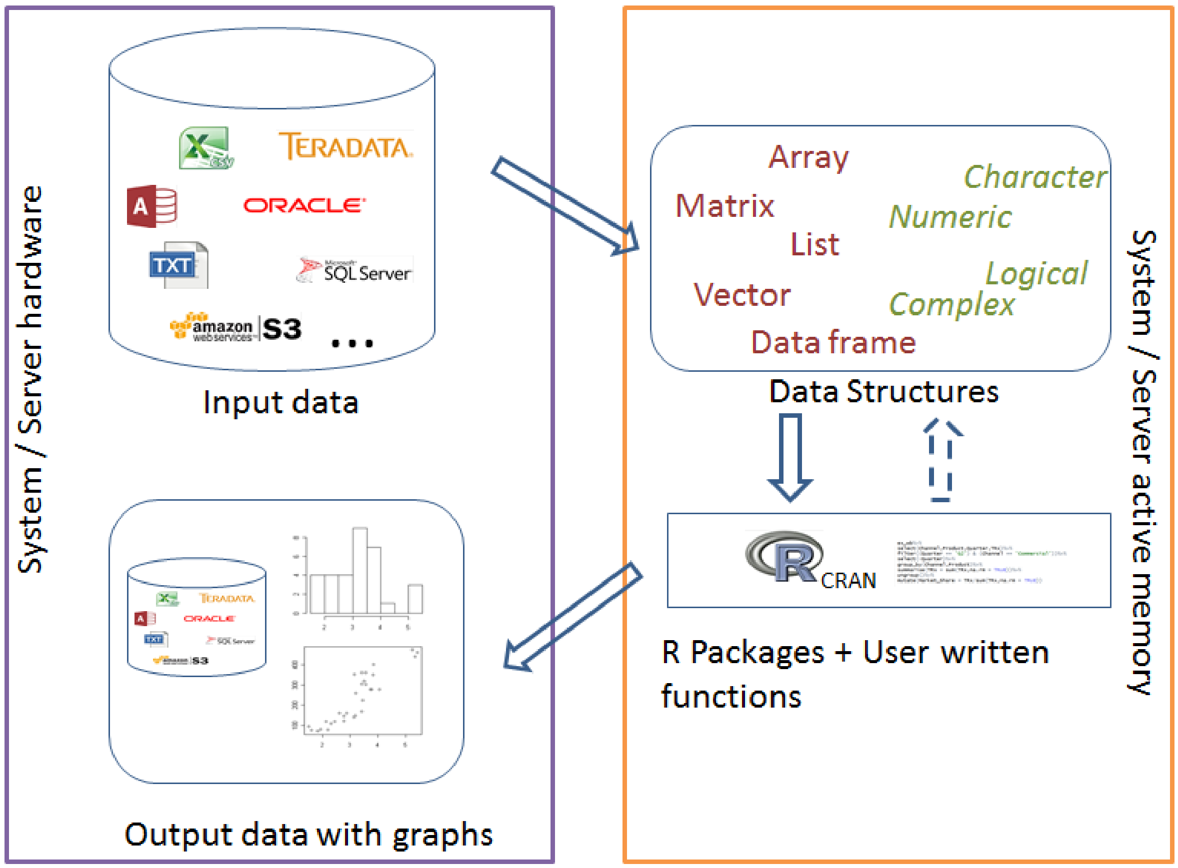




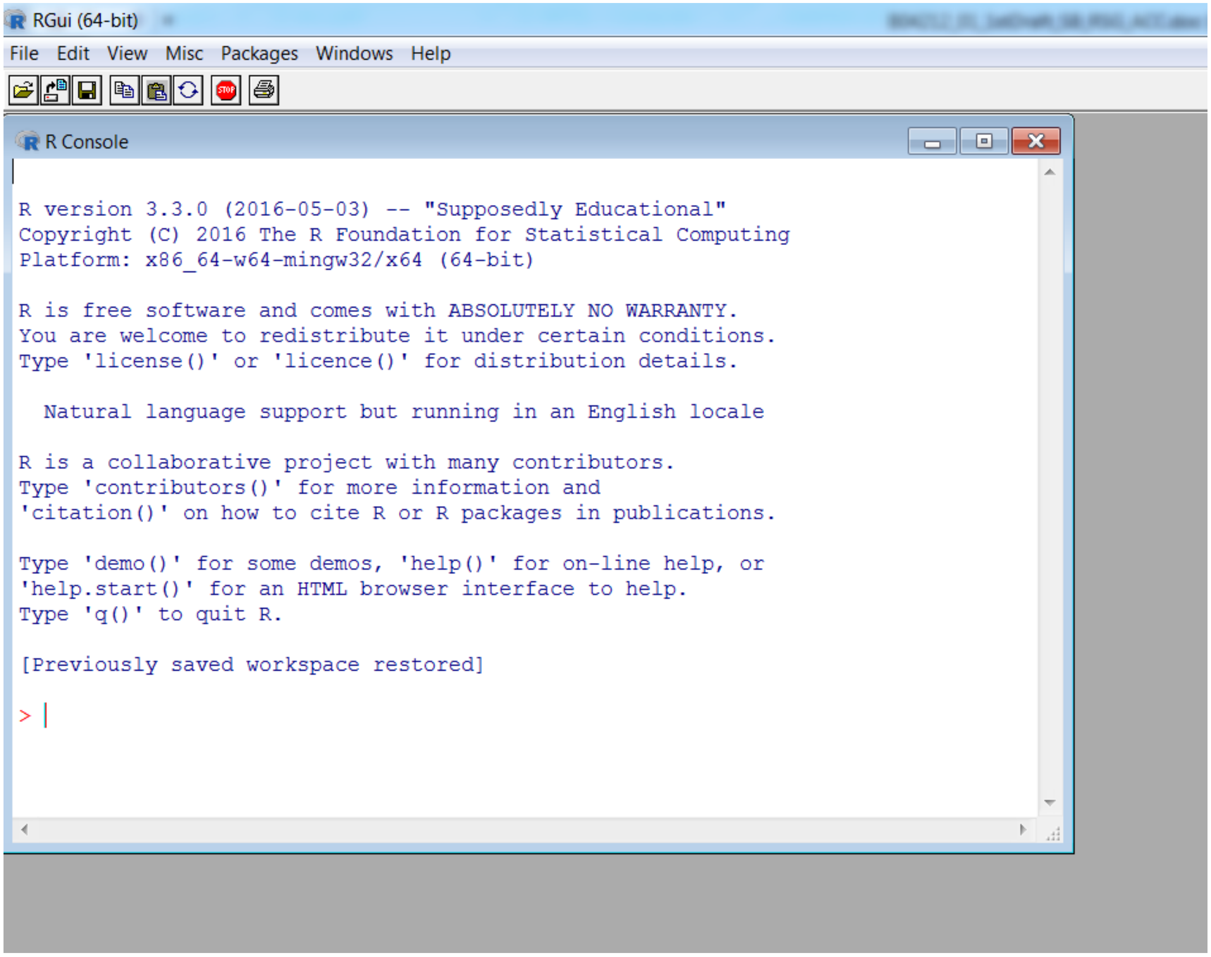


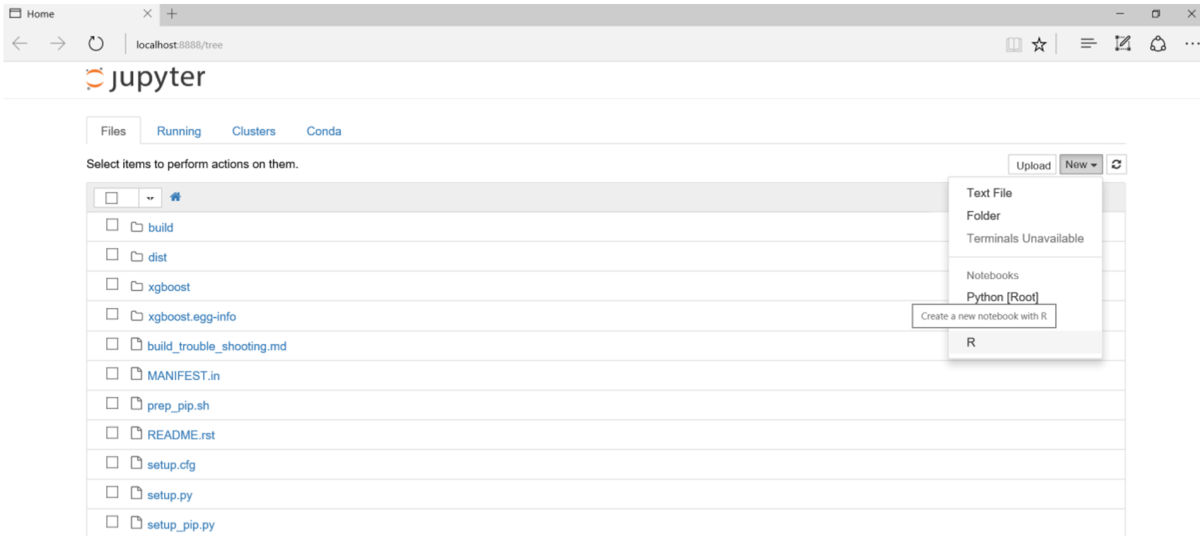
a) Stack Implementation

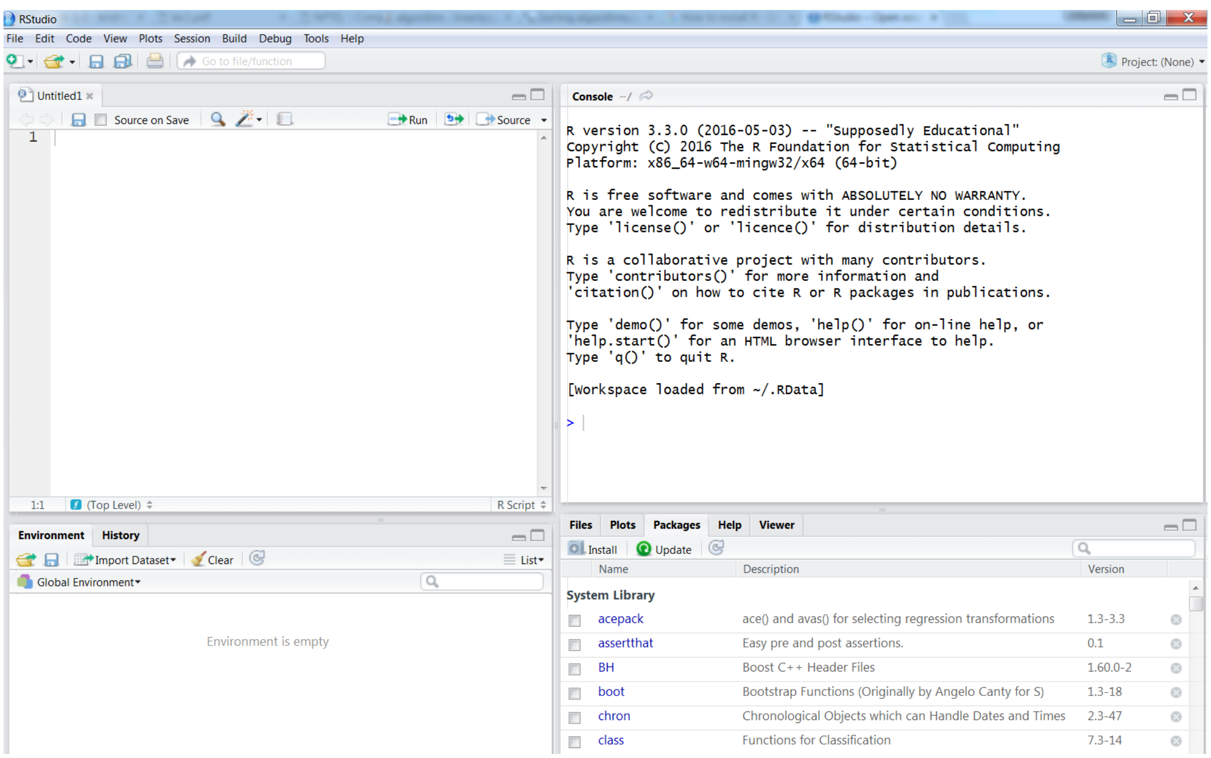
b) Queue Implementation



System / Server hardware
TERADATA
ORACLE
TXT
SQL Server
amazon web services S3
...
Input data
Output data with graphs
Array
Character
Matrix
Numeric
List
Logical
Vector
Complex
Data frame
Data Structures
CRAN
R Packages + User written functions
System / Server active memory


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Previously saved workspace restored]
>


Home
localhost:8888/tree
jupyter
Files
Running
Clusters
Conda
Select items to perform actions on them.
Upload
New
Text File
Folder
Terminals Unavailable
Notebooks
Python [Root]
Create a new notebook with R
R
build
dist
xgboost
xgboost.egg-info
build_trouble_shooting.md
MANIFEST.in
prep_pip.sh
README.rst
setup.cfg
setup.py
setup_pip.py


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Project: (None)
Untitled1
Source on Save
Run
Source
1:1
(Top Level)
R Script
Console ~/
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Workspace loaded from ~/.RData]
Environment
History
Import Dataset
Clear
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Install
Update
Name
Description
Version
System Library
acepack
ace() and avas() for selecting regression transformations
1.3-3.3
assertthat
Easy pre and post assertions.
0.1
BH
Boost C++ Header Files
1.60.0-2
boot
Bootstrap Functions (Originally by Angelo Canty for S)
1.3-18
chron
Chronological Objects which can Handle Dates and Times
2.3-47
class
Functions for Classification
7.3-14

	Homogeneous	Heterogeneous
1-D	Atomic vector	List*
2-D	Matrix	Data Frame
n-D	Array	

*List can be converted into *n-D* by composite usage

Operators	Explanations
+	Addition
-	Subtraction
*	Multiplication
/	Division
** or ^	Exponentiation
%%	Modulus
%/%	Integer Quotient

Operators	Explanations
==	Exact equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

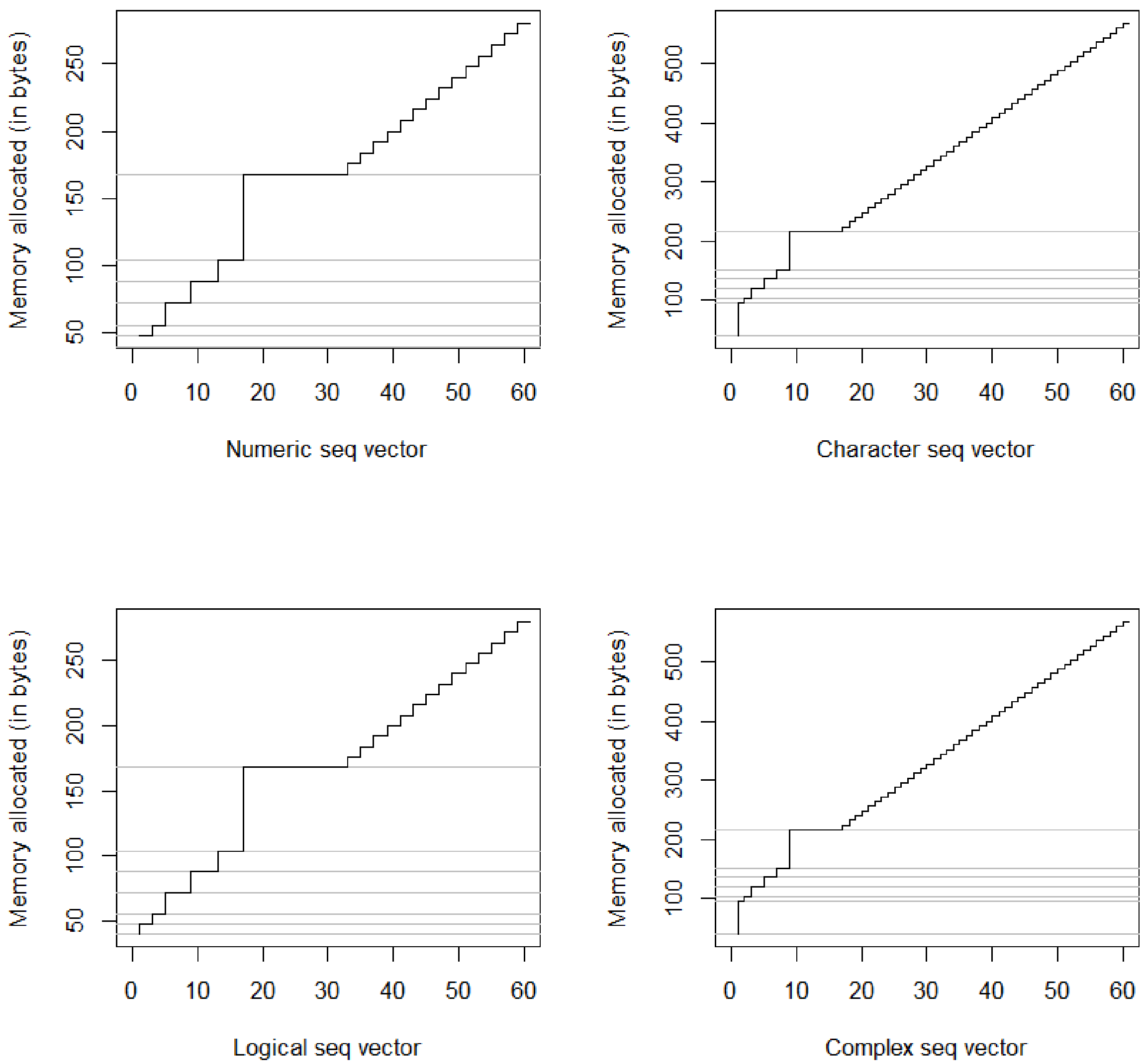
Function	Input data type	Output data type
apply	dataframe or matrix or array (with margins)	vector, matrix, array, list
lapply	vector, list, variables in dataframe or matrix	list
sapply	vector, list, variables in dataframe or matrix	matrix, vector, list
mapply (multivariate sapply)	vector, list, variables in dataframe or matrix	matrix, vector, list
tapply	ragged array	array
rapply	vector, list, variables in	list

```
> dat<- list(c(4, 2, 6, 1, 5), c("P", "S", "N", "K", "K"))
> tapply(1:5, dat, sum)
   K  N  P  S
1  4 NA NA NA
2 NA NA NA  2
4 NA NA  1 NA
5  5 NA NA NA
6 NA  3 NA NA
```

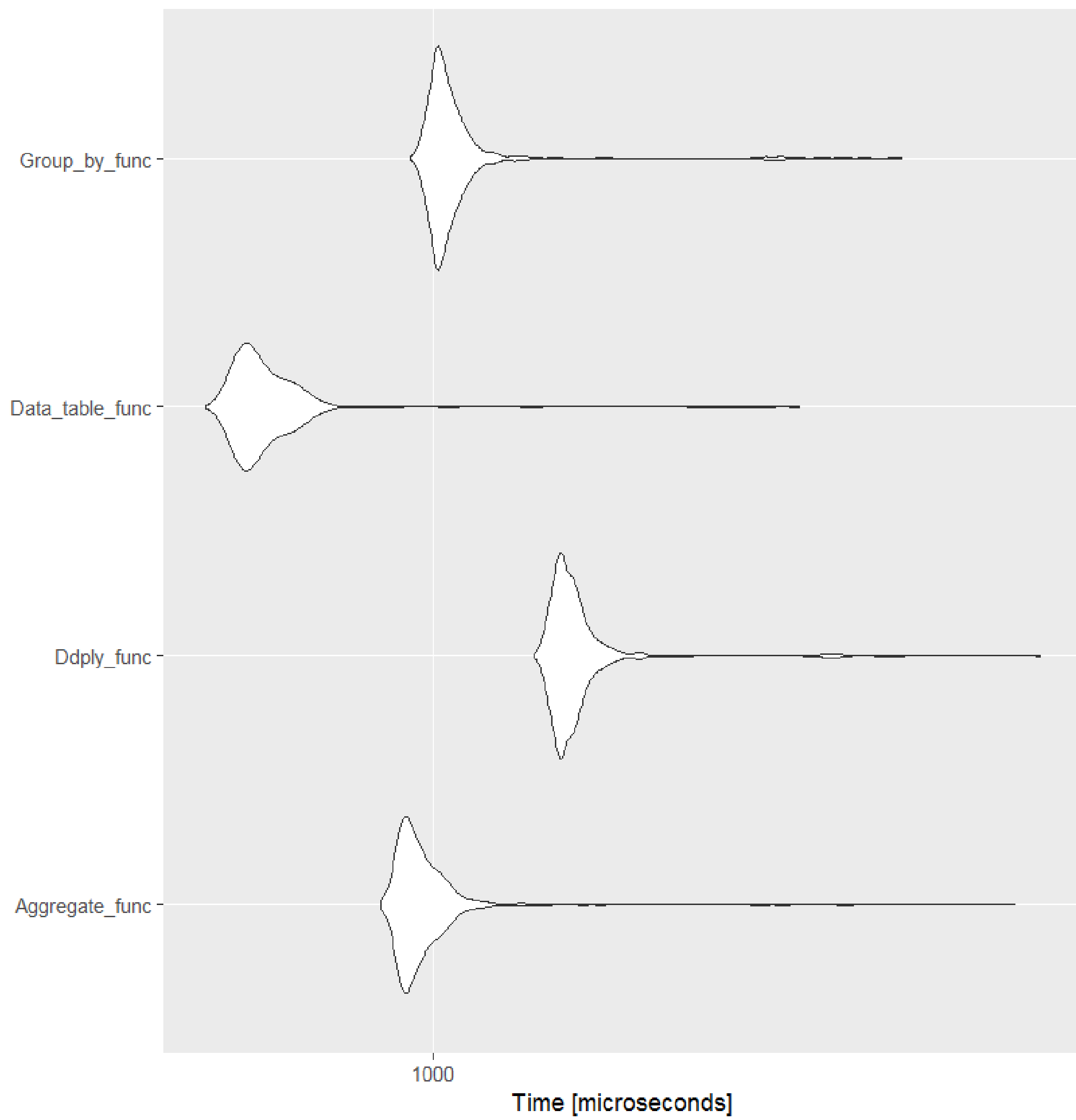
Chapter 2: Algorithm Analysis

Size of Input data *n*	Double log form *loglog n*	Single log form *log n*	Linear form *n*	N times log form *n*log n*	Quadratic form n^2	Cubic form n^3	Exponential form 2^n
4	1	2	2^2	2^3	2^4	2^6	2^4
16	2	4	2^4	2^6	2^8	2^{12}	2^{16}
256	3	8	2^8	2^{11}	2^{16}	2^{24}	2^{256}
512	~3.2	9	2^9	$\sim 2^{12}$	2^{18}	2^{27}	2^{512}
1,024	~3.3	10	2^{10}	2^{13}	2^{20}	2^{30}	2^{1024}
5,000	~3.62	~12.28	2^{12}	$\sim 2^{16}$	$\sim 2^{24}$	$\sim 2^{36}$	2^{5000}
10,000	~3.73	~13.28	2^{13}	$\sim 2^{17}$	$\sim 2^{26}$	$\sim 2^{39}$	2^{10000}
50,000	~3.96	~15.61	2^{16}	$\sim 2^{20}$	$\sim 2^{32}$	$\sim 2^{49}$	2^{50000}
100,000	~4.05	~16.61	2^{17}	$\sim 2^{21}$	$\sim 2^{34}$	$\sim 2^{51}$	2^{100000}
1000,000	~4.31	~19.93	2^{20}	$\sim 2^{24}$	$\sim 2^{40}$	$\sim 2^{60}$	$2^{1000000}$

S. No.	Data Type	Package	Memory allocation (Bytes)
1	Numeric	base	40
2	Character	base	40
3	Logical	base	40
4	Complex	base	40
5	Vector	base	40
6	List	base	40
7	Matrix	base	208
8	Data Frame	base	560
9	Data Table	data.table	846

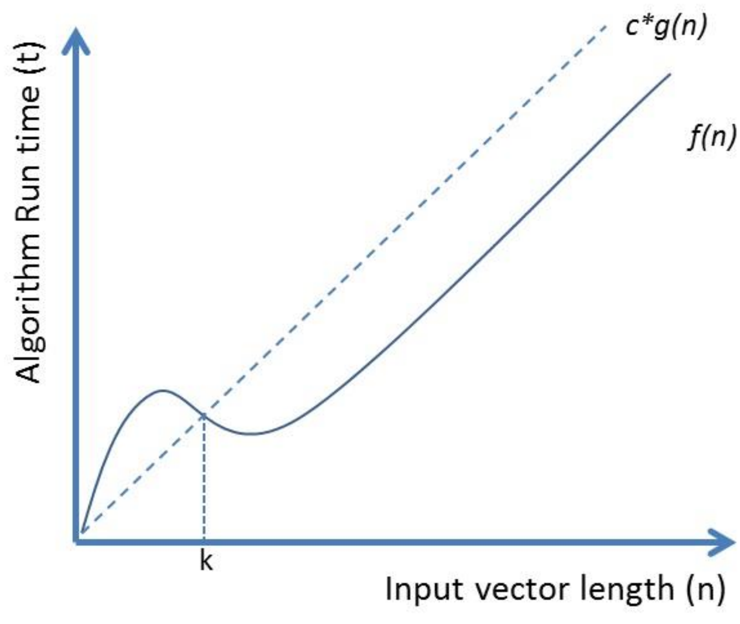

Memory allocated (in bytes)
50
100
150
200
250
0
10
20
30
40
50
60
Numeric seq vector
Memory allocated (in bytes)
100
200
300
400
500
0
10
20
30
40
50
60
Character seq vector
Memory allocated (in bytes)
50
100
150
200
250
0
10
20
30
40
50
60
Logical seq vector
Memory allocated (in bytes)
100
200
300
400
500
0
10
20
30
40
50
60
Complex seq vector

```
> MB_res
Unit: microseconds
            expr      min        lq      mean   median        uq      max neval
  Aggregate_func  851.489  913.8015 1001.9007  944.775 1000.4905 6094.209  1000
      Ddply_func 1370.519 1475.1685 1579.6123 1517.322 1575.7855 6598.578  1000
 Data_table_func  493.739  552.7540  610.7791  577.495  621.6635 3125.179  1000
   Group_by_func  932.129 1008.5540 1095.4193 1033.113 1076.1825 4279.435  1000
```

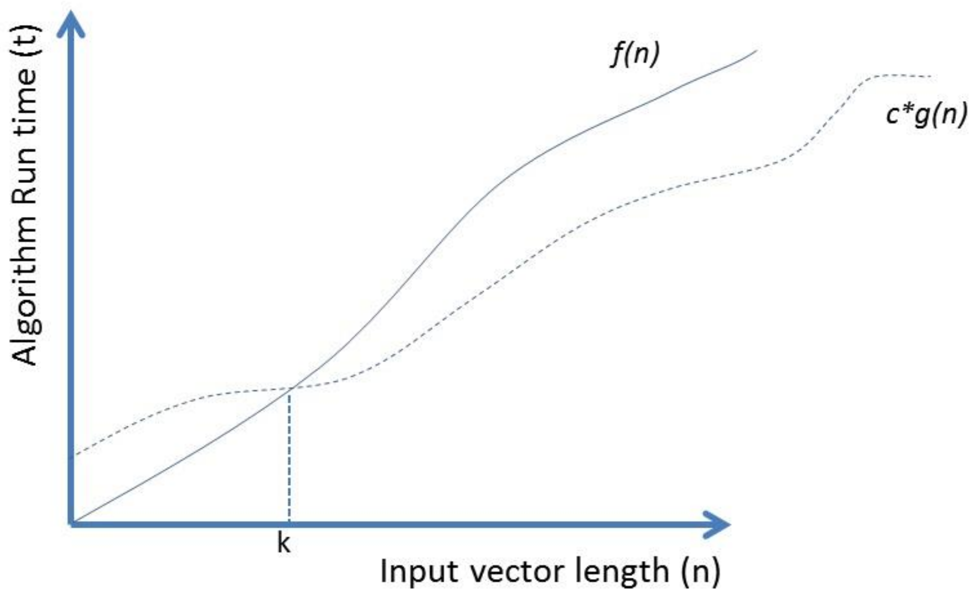


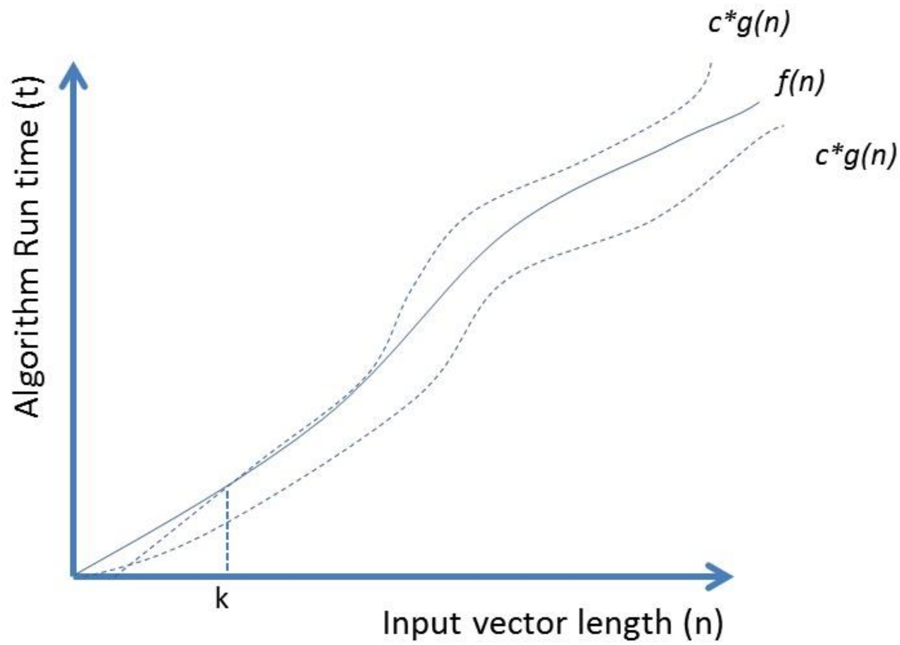
Group_by_func
Data_table_func
Ddply_func
Aggregate_func
1000
Time [microseconds]

Functional form of growth rate f(n)	~Size of dataframe to perform 100,000 operations in computer A (n_1)	~Size of dataframe to perform 1,000,000 operations in computer B (n_2)	Methodological change of n_1 towards n_2	Ratio of n_2 upon n_1
$k * n$	100,000 / k	1,000,000/k	$n_2 = 10 * n_1$	10
$k*Log_{10}(n)$	$10^{100,000/k}$	$10^{1,000,000/k}$	$n_2 = \sqrt[k]{10}\, n_1$	$\sqrt[k]{10}$
$k*nLog_{10}(n)$	$10^{100,000/k}$> $n_1 > \sqrt{(100,000/k)}$	$10^{1,000,000/k}$> $n_2 > \sqrt{(1,000,000/k)}$	$\sqrt{10} * n1 < n_2 < k * n_1$	$\sqrt{10}$ to 10
$k*n^2$	$\sqrt{(100,000/k)}$	$\sqrt{(1,000,000/k)}$	$n_2 = \sqrt{10} * n_1$	$\sqrt{10}$
$k*n^3$	$\sqrt[3]{100,000/k)}$	$\sqrt[3]{1,000,000/k)}$	$n_2 = \sqrt[3]{10} * n_1$	$\sqrt[3]{10}$
$k*2^n$	$Log_2(100,000/k)$	$Log_2(1,000,000/k)$	$n_2 = Log_{10^\wedge 5/k}(10^6/k) * n_1$	$Log_{10^\wedge 5/k}(10^6/k)$



Type of growth order	Representation using Big O notation
Constant	O(1)
Linear	O(n)
Quadratic	$O(n^2)$
i^{th} order	$O(n^i)$
Logarithmic	$O(\log_2 n)$
$n \log_2(n)$	$O(n \log_2 n)$
Polynomial of order *i*	$n^{O(i)}$
Exponential	$2^{O(n)}$ or $O(2^n)$ *





Property	Rule definition	Interpretation
Transitive	If f(n) = O(g(n)) and g(n) = O(h(n)), then f(n) = O(h(n))	Upper bound of an upper bound is always an upper bound to any growth rate function f(n).
Constants	If f(n) = O(c*g(n)), then f(n) = O(g(n)) for any constant c>0	Constants can be ignored while determining simplest forms for any growth rate function f(n).
Sequence	If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(\max(g_1(n),g_2(n)))$	The most costly part of the simplest forms is considered when two parts of a growth rate functions run in sequence.
Loop	If f(n) = O(g(n)) then $n*f(n) = n*O(g_1(n))$ where n is number of repeat iterations within a loop	The cost associated with each iteration can be simply added when a growth rate function runs within a loop.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{\lim_{n\to\infty} f(n)}{\lim_{n\to\infty} g(n)}$$

Condition	Observation	Comparison in terms of simplest form
If limit tends to infinity	Then, f(n) has faster growth rate than g(n)	$f(n) = \Omega(g(n))$
If limit tends to zero	Then, f(n) has slower growth rate than g(n)	$f(n) = O(g(n))$
If limits tends to a constant greater than zero	Then, f(n) has a comparable growth rate as g(n)	$f(n) = \theta(g(n))$

Line wise	System run time	Simplifying rule	Asymptotes (big Theta notation)
a <- 0	Constant	Constant	$\theta(1)$
for(i in 1:n)	---		
a <- a + i	Constant (repeats n times)	Loop	$\theta(n)$

Line wise	System run time	Simplifying rule	Asymptotes (big Theta notation)
a <- 1 i <- 1 b <- list()	Constant	Constant	$\theta(c_1) \sim \theta(1)$
while(i<=n)	---		

{
a <- a + i
i<- i+1
}
	Constant (repeats n times)	Loop	$\theta(c_2 * n) \sim \theta(n)$

for(j in 1:i) for(k in 1:i)	---		

{
b[[j]] <- a+j*k
}
	Constant
	(For each j, k iterates n times)
		Nested Loop	$\theta(c_3 * n^2) \sim \theta(n^2)$

Line wise	System run time	Simplifying rule	Asymptotes (big Theta notation)
a <- 1	Constant	Constant	$\theta(c_1) \sim \theta(1)$

for(i in 1:n)
{. . .
}
	--- 			

if(i <= n/2)
{
for(j in 1:i)
a <- a+i
}
	Constant (repeats n(n+3)/8 times)	Nested Loops when if condition is True	$\theta(c_2 * n(n+3)/8) \sim \theta(n^2)$

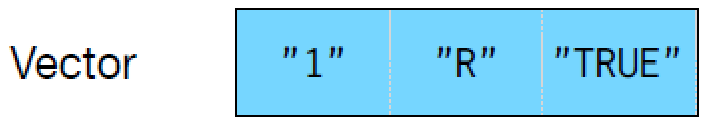
else{
a <- a*i
}
	Constant (repeats n/2 times)	Simple Loop when if condition is False	$\theta(c_3 * n) \sim \theta(n)$

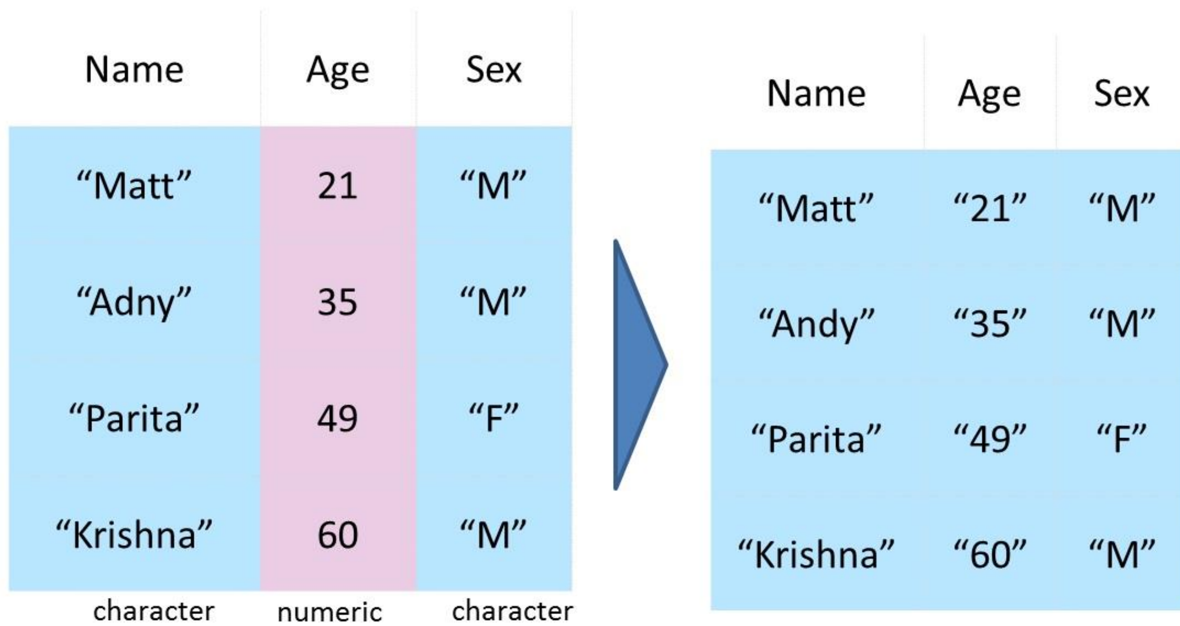
Line wise	System run time	Simplifying rule	Asymptotes (big Theta notation)
fact_n <- 1	Constant	Constant	$\theta(c_1) \sim \theta(1)$

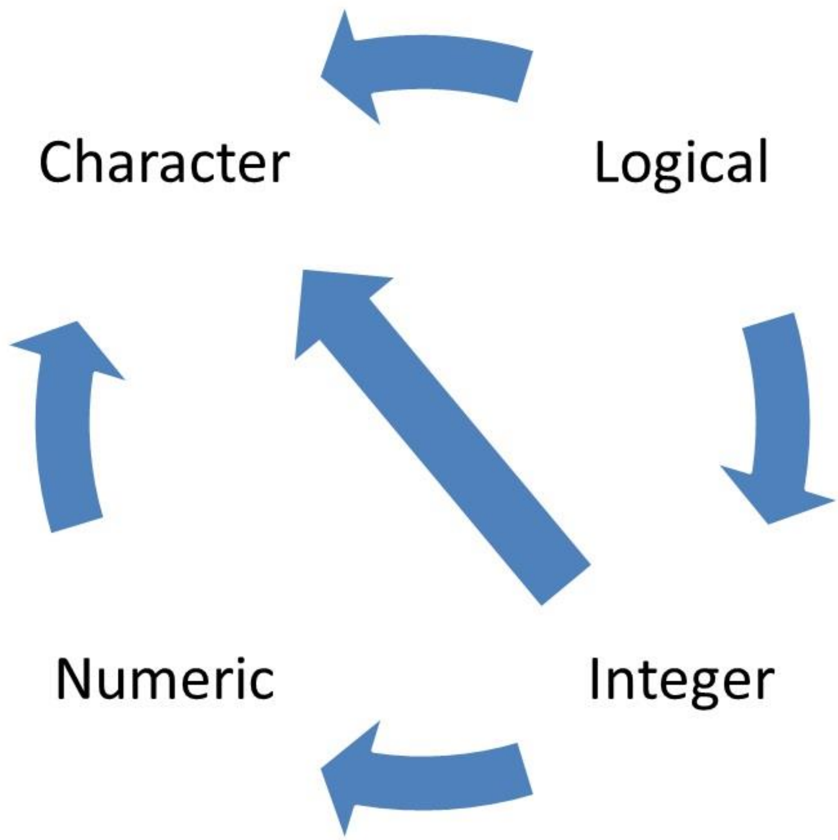
for(i in 2:n)
{. . .
}
	--- 			

fact_n <- fact_n * i	Constant (repeats n times)	Loop	$\theta(c_2 * n) \sim \theta(n)$

Chapter 3: Linked Lists

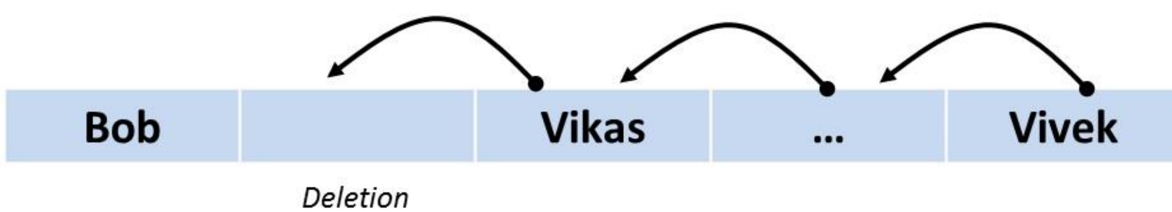





Character
Logical
Numeric
Integer

Operators in R	Mode Conversion		Examples
	From	To	
as.numeric	Character	Numeric	"1", "2.5","3" ---> 1, 2.5, 3
			"A","B","C" ---> NA, NA, NA
	Logical	Numeric	TRUE ---> 1
			FALSE ---> 0
as.character	Numeric	Character	1, 2, 3, 4 ---> "1", "2", "3", "4"
	Logical	Character	TRUE ---> "TRUE"
			FALSE ---> "FALSE
as.logical	Numeric	Logical	0 ---> FALSE
			Non-Zero ---> TRUE
	Character	Logical	"F", "FALSE" ---> FALSE
			"T", "TRUE" ---> TRUE
			Others ---> NA

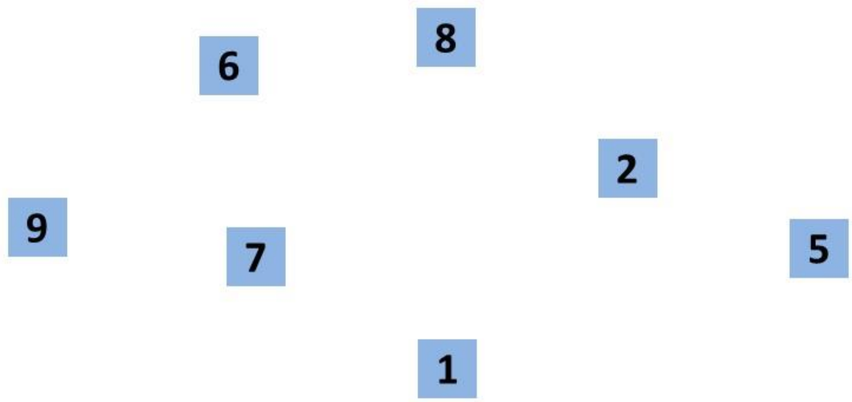
typeof	mode	storage.mode
logical	logical	logical
integer	numeric	Integer
double	Numeric	double
complex	complex	complex
character	character	character
raw	raw	raw



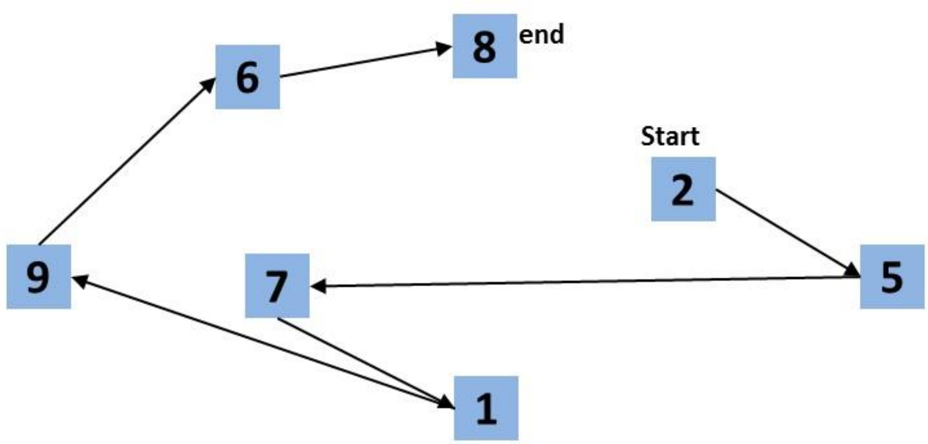
Methods	Generic function
Print any object	*print()*
Extract summary of any object	*summary()*
Plotting multiple objects	*plot()*

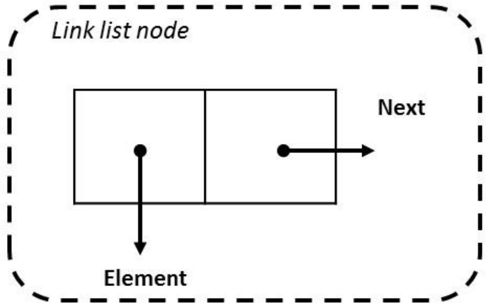
Properties	S3	S4	R5 (Reference class)
Identify class of an object	pryr::otype()	pryr::otype() or isS4()	(is(x,"refClass")).pryr::otype()
Identify class of a generic function and method	pryr::ftype()	pryr::ftype() or isS4()	(is(x,"refClass")).pryr::ftype()
Define classes using	Not applicable	setClass()	setRefClass()
Create new objects using	Class attributes	new()	Generator functions
Access attributes using	$	@	$
Methods belong to	Generic functions	Generic functions	Classes
Follows copy on modify semantics	Yes	Yes	No

(a) Contiguous memory allocation

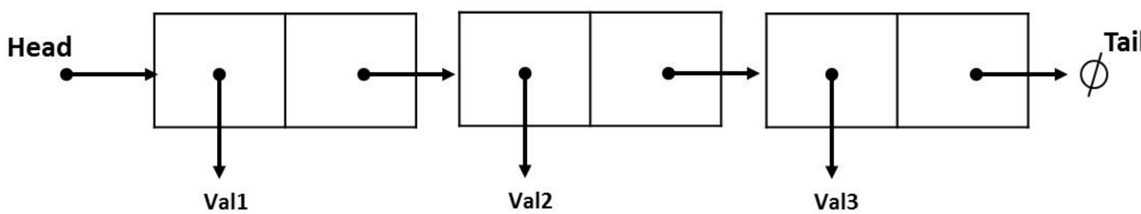


(b) Non-contiguous memory allocation





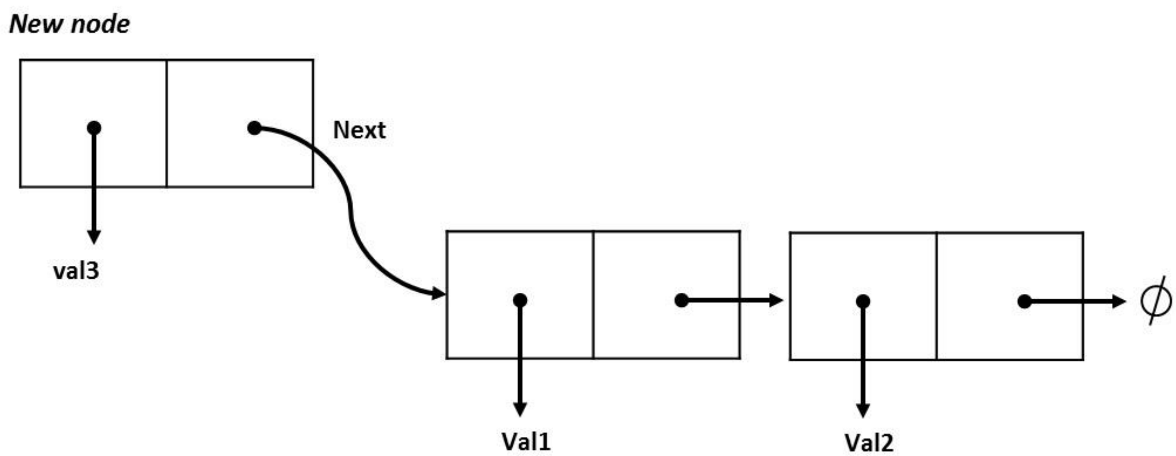
a) Example of node of link list

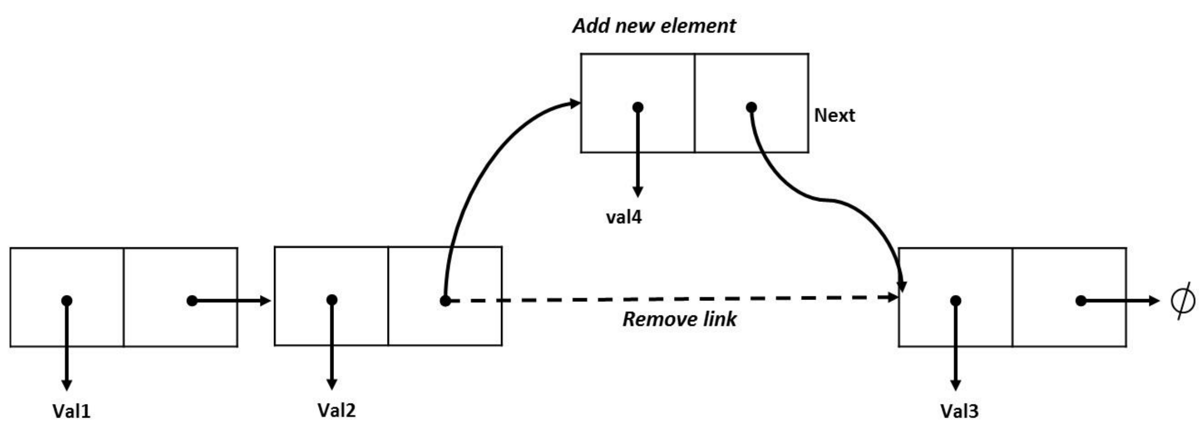


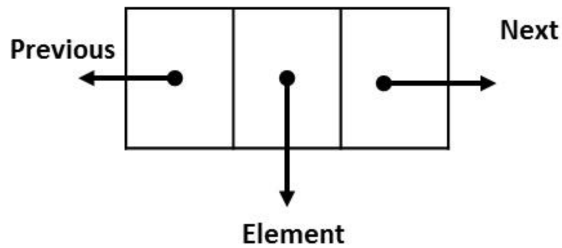
b) Example of link list

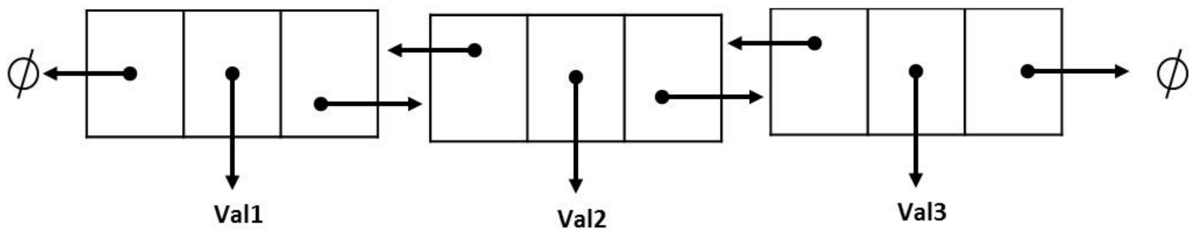
S. No.	Operation	Input	Output
1	Create new empty list	None	Empty list
2	Boolean Check if link list is empty	List	Return Boolean value {True, False}
3	Get size of list	None	Return size as an integer
4	Add an item to existing list	Item to be added	Modified list
5	Remove an item from existing list	Item to be deleted	Modified list
6	Searches for an item in the list	Item to be searched	Return Boolean value {True, False}

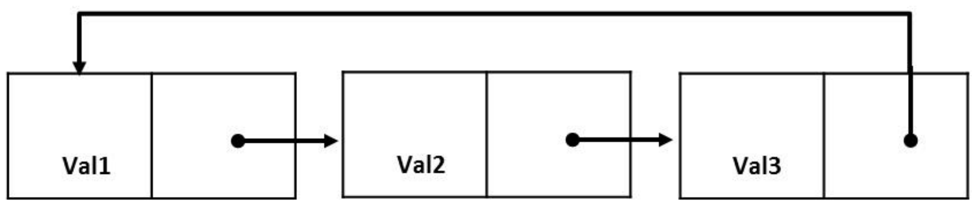
$$< e_n >_{n \in N}$$


New node
Next
val3
Val1
Val2
∅

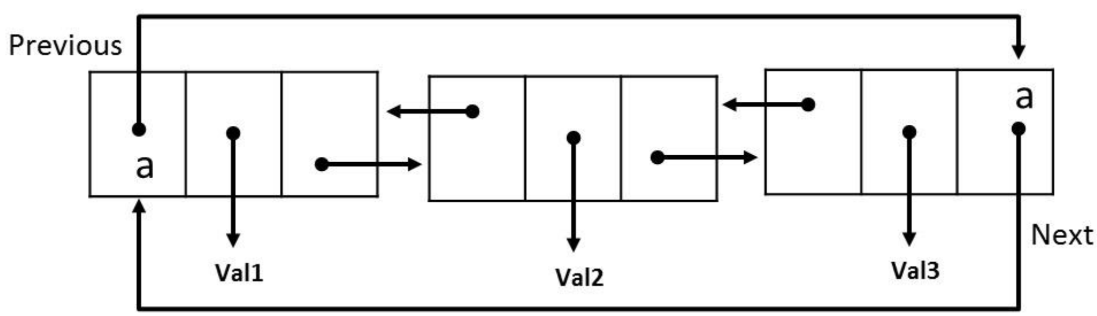

Add new element
Next
val4
Remove link
Val1
Val2
Val3
∅


Previous
Next
Element

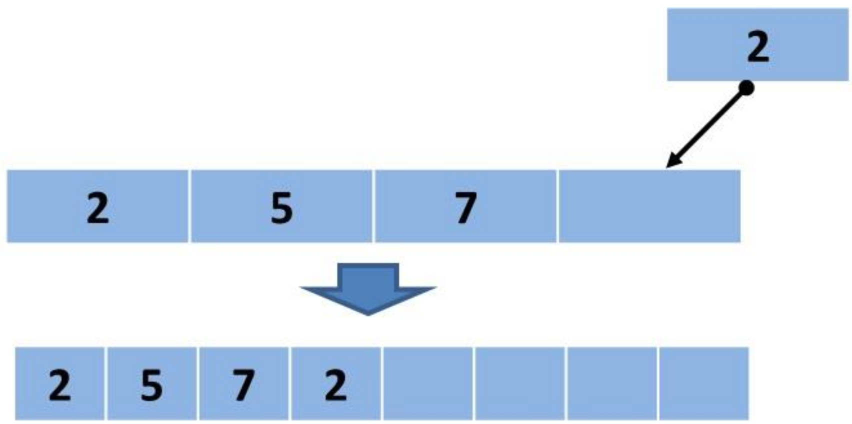




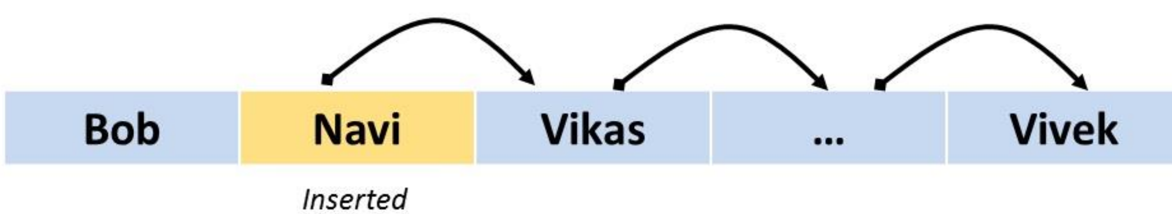
a) Circular singly linear link list



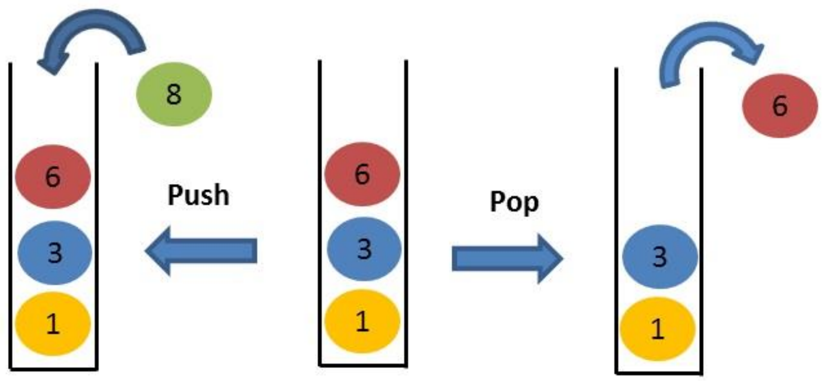
b) Circular doubly link list



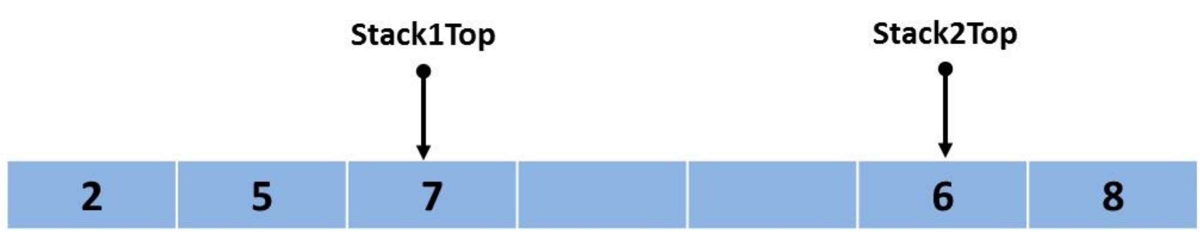
S. No.	Operation	Input	Output
1	Create new empty array list	None	Empty array list
2	Get size of array list	None	Return size as an integer
3	Add an item to existing array list and expand if it's filled	Item to be added	Modified list
4	Remove an item from existing array list based on position	position to be deleted	Modified list
5	Searches for an item in the array list	Item to be searched	Return Boolean value {True, False}

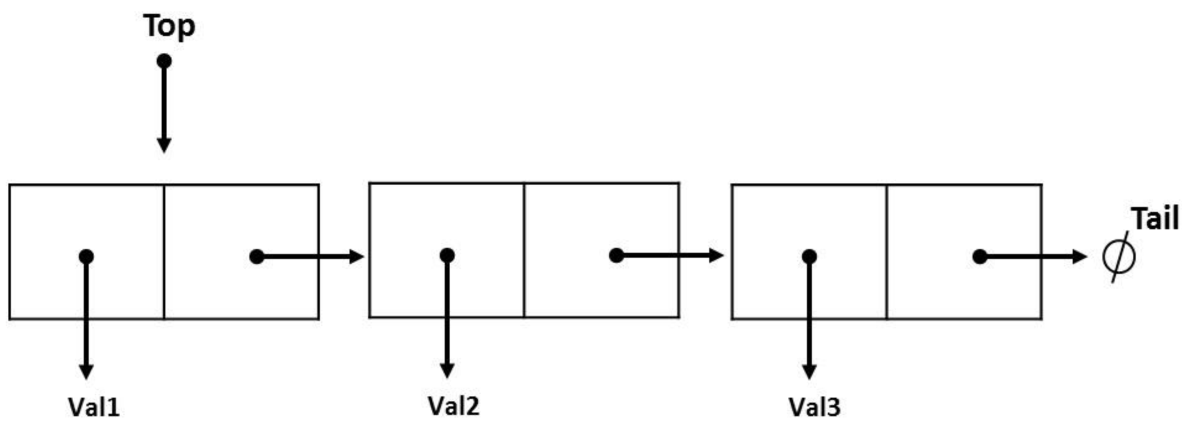


Chapter 4: Stacks and Queues

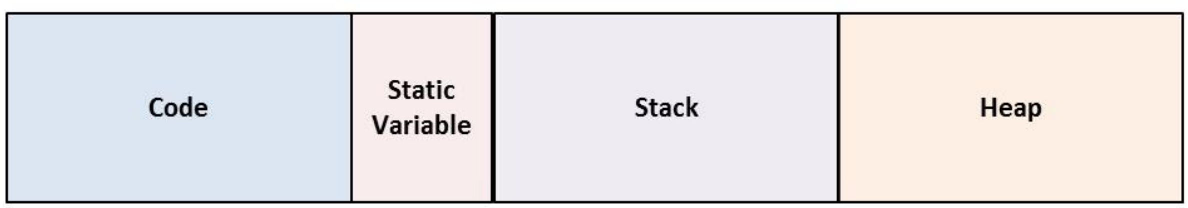


S. No.	Operation	Input	Output
1	Create new empty stack	None	Empty stack
2	Check if stack is empty	stack	Return Boolean value {True, False}
3	PUSH an element to the stack	Item to be PUSHED	Modified Stack
4	POP an element from stack	None	Modified stack
5	Size of stack	stack	Return size of stack
6	Top value in the stack	Stack	Return top value of stack

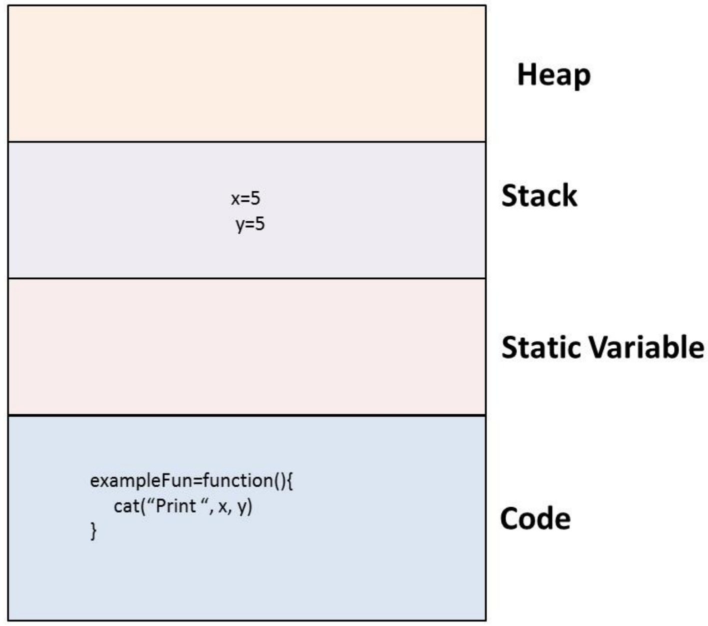


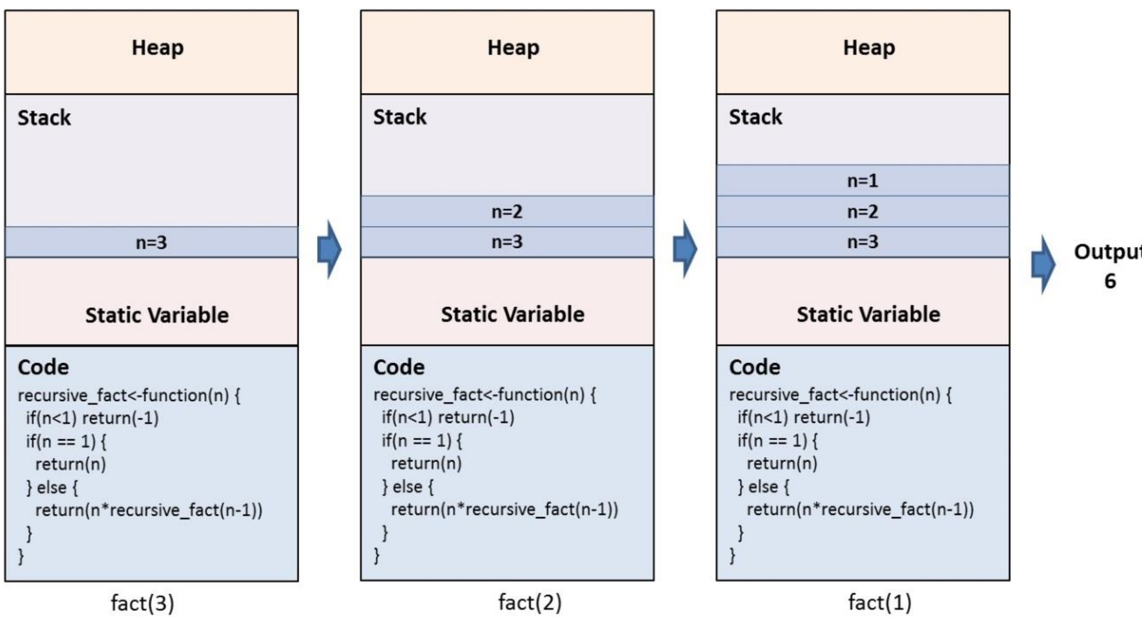



```
Reference class object of class "Linkstack"
Field "Lsize":
[1] 1
Field "Lstacktop":
<environment: 0x00000000405fc248>
```

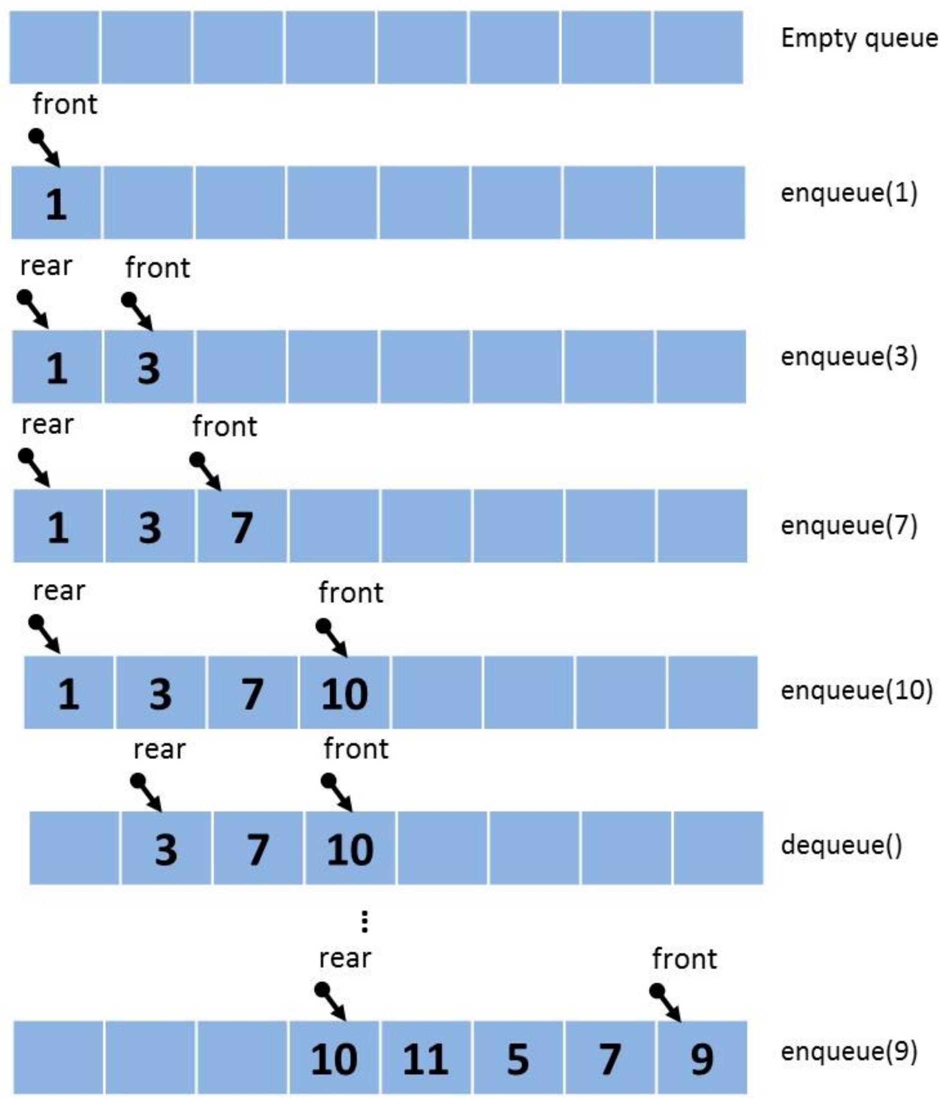


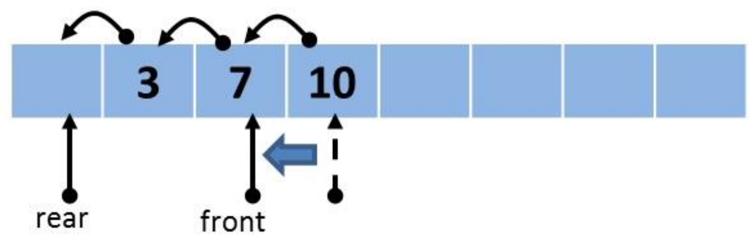
```
exampleFun = function(){
   x=5
   y=5
    cat("Print ", x, y)
}
```

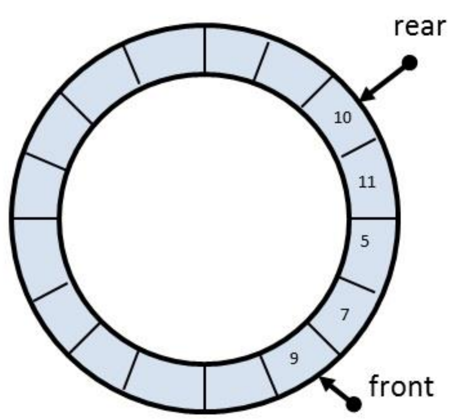


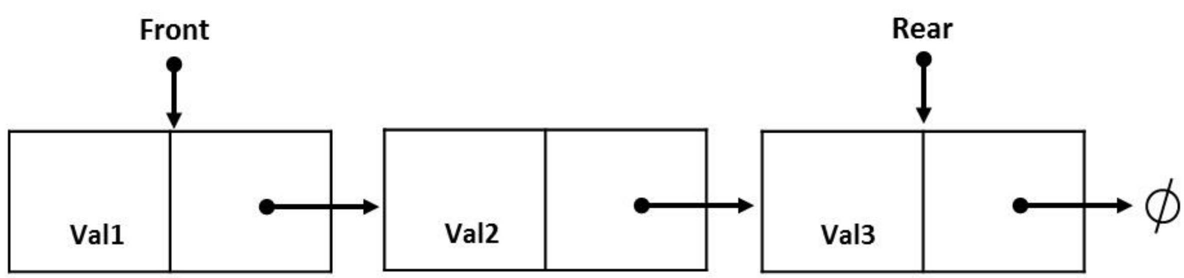


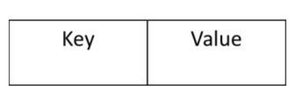
S. No.	Operation	Input	Output
1	Create new empty queue	None	Empty queue
2	Add an element to the queue	Item to be added	Modified queue
3	Delete an element from queue	None	Modified queue
4	Size of queue	Queue	Return size of queue
5	Check if queue is empty	Queue	Return Boolean value {True, False}


Empty queue
front
1
enqueue(1)
rear
front
1
3
enqueue(3)
rear
front
1
3
7
enqueue(7)
rear
front
1
3
7
10
enqueue(10)
rear
front
3
7
10
dequeue()
rear
front
10
11
5
7
9
enqueue(9)


3
7
10
rear
front

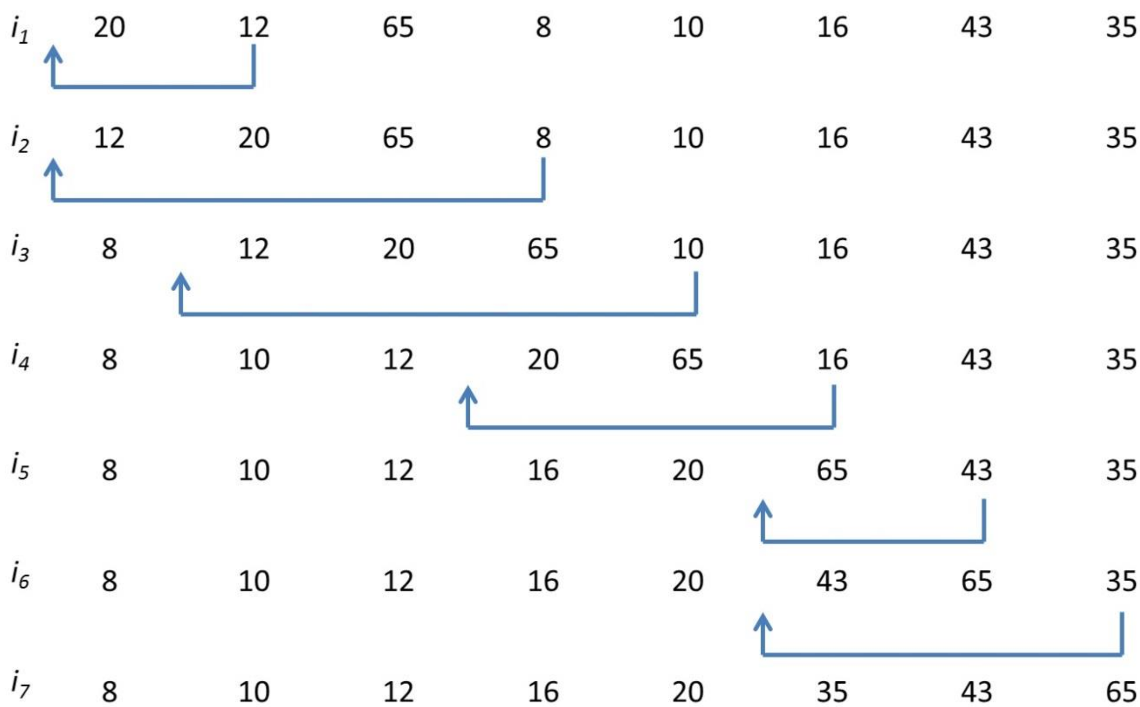

rear
10
11
5
7
9
front


Front
Rear
Val1
Val2
Val3
∅


Key
Value

S. No.	Operation	Input	Output
1	Initialization dictionary	Dictionary	Empty dictionary
2	Add an element	Key and value pair	Updated dictionary
3	Delete an element	Key	Updated dictionary
4	Size	None	Size of dictionary
5	Find value based on key	Key	Boolean {TRUE, FALSE}

Chapter 5: Sorting Algorithms

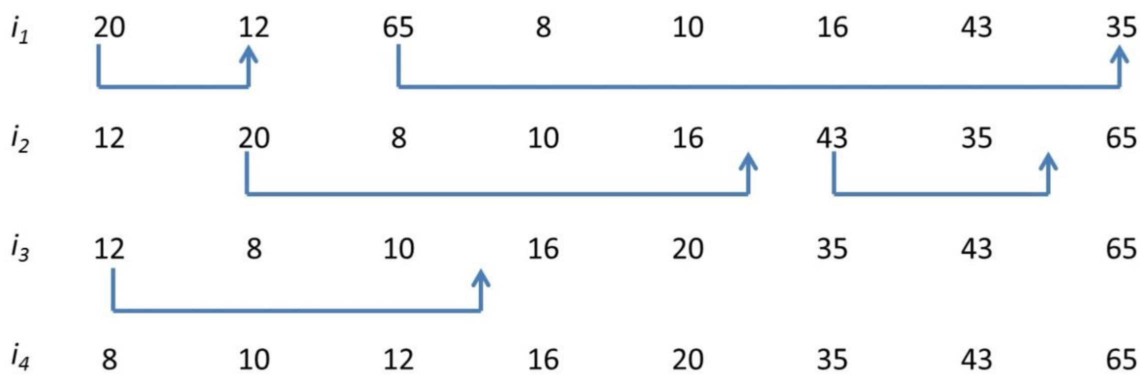


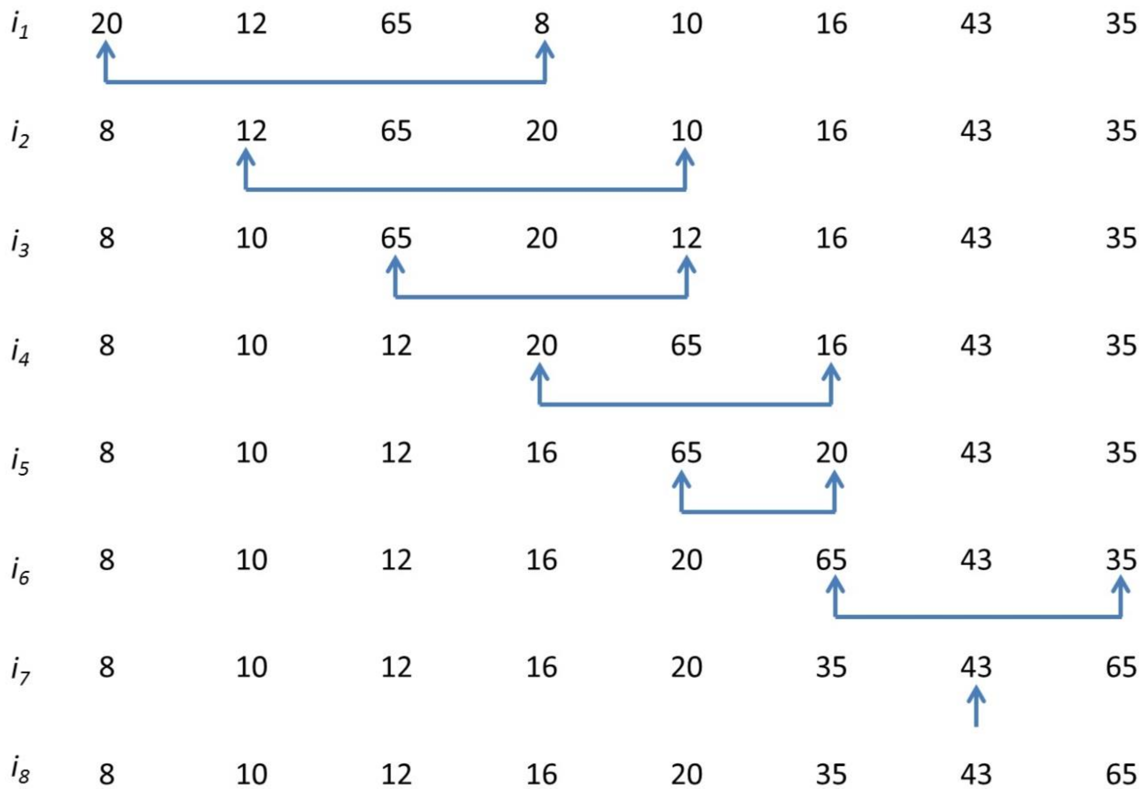
$$\sum_{i=2}^{n} i = \frac{(n-1)(n)}{2} \sim \theta\left(n^2\right)$$

$$\sum_{i=2}^{n-1} 1 = n = \theta(n)$$

$$\left(\frac{n}{2} - 1\right)$$

$$\left(\frac{(n-2)(n)}{8}\right)$$





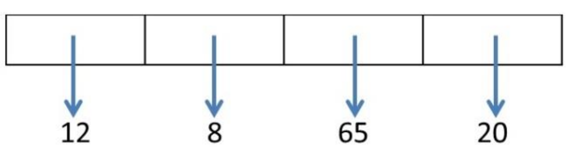


Fig (a)

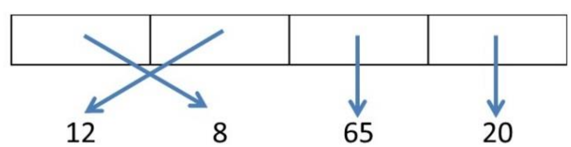
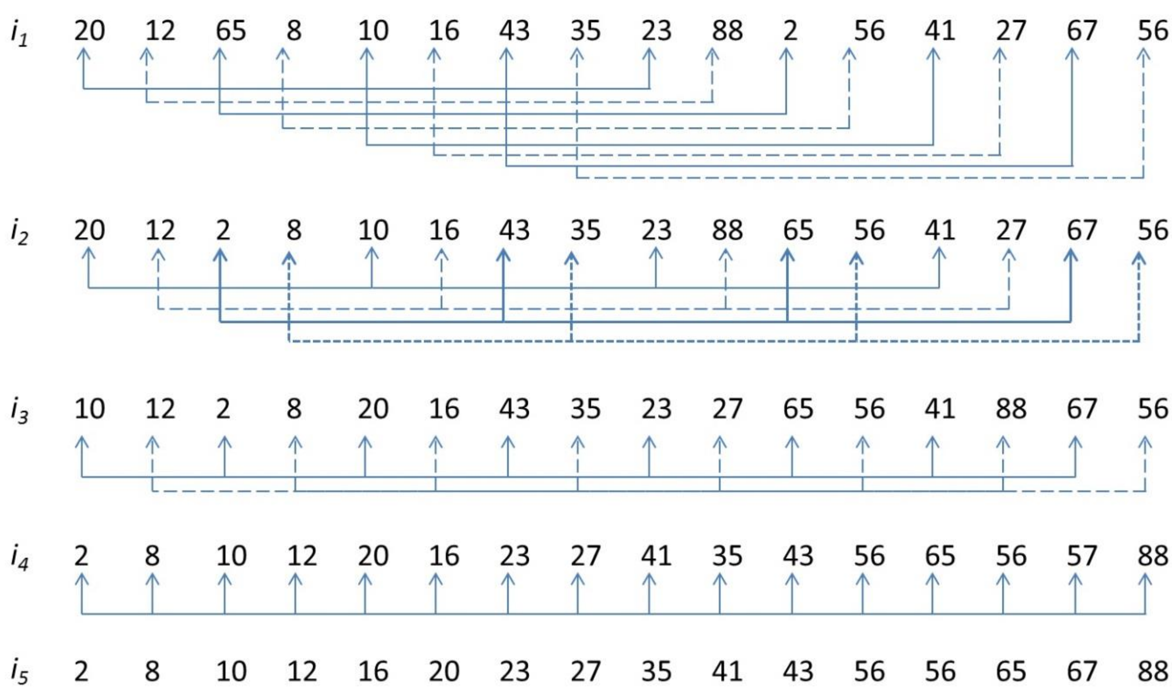


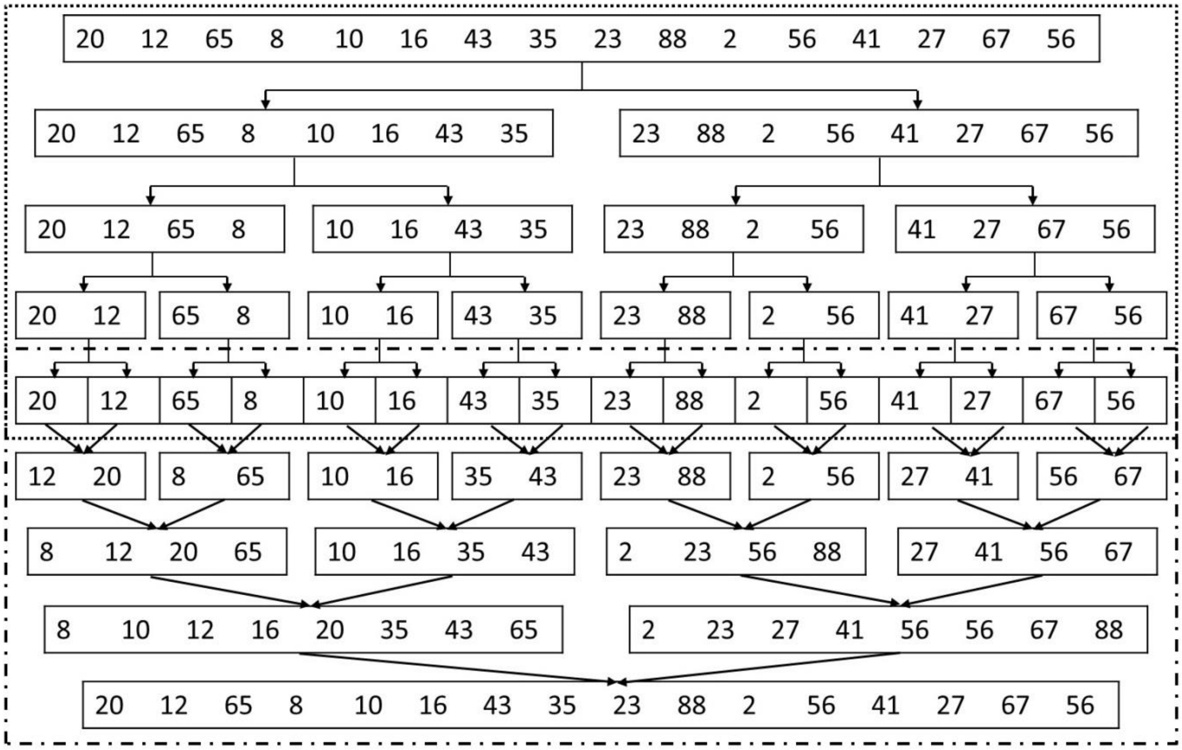
Fig (b)

$$\frac{n\left(n-1\right)}{2}$$

$$\frac{n(n-1)}{4}$$

		Insertion sort	Bubble sort	Selection sort
Number of Comparisons	Best case	Θ(n)	$\Theta(n^2)$	$\Theta(n^2)$
	Average case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
	Worst case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Number of Swaps	Best case	0	0	Θ(n)
	Average case	$\Theta(n^2)$	$\Theta(n^2)$	Θ(n)
	Worst case	$\Theta(n^2)$	$\Theta(n^2)$	Θ(n)




20 12 65 8 10 16 43 35 23 88 2 56 41 27 67 56
20 12 65 8 10 16 43 35
23 88 2 56 41 27 67 56
20 12 65 8
10 16 43 35
23 88 2 56
41 27 67 56
20 12
65 8
10 16
43 35
23 88
2 56
41 27
67 56
20 12 65 8 10 16 43 35 23 88 2 56 41 27 67 56
12 20
8 65
10 16
35 43
23 88
2 56
27 41
56 67
8 12 20 65
10 16 35 43
2 23 56 88
27 41 56 67
8 10 12 16 20 35 43 65
2 23 27 41 56 56 67 88
20 12 65 8 10 16 43 35 23 88 2 56 41 27 67 56

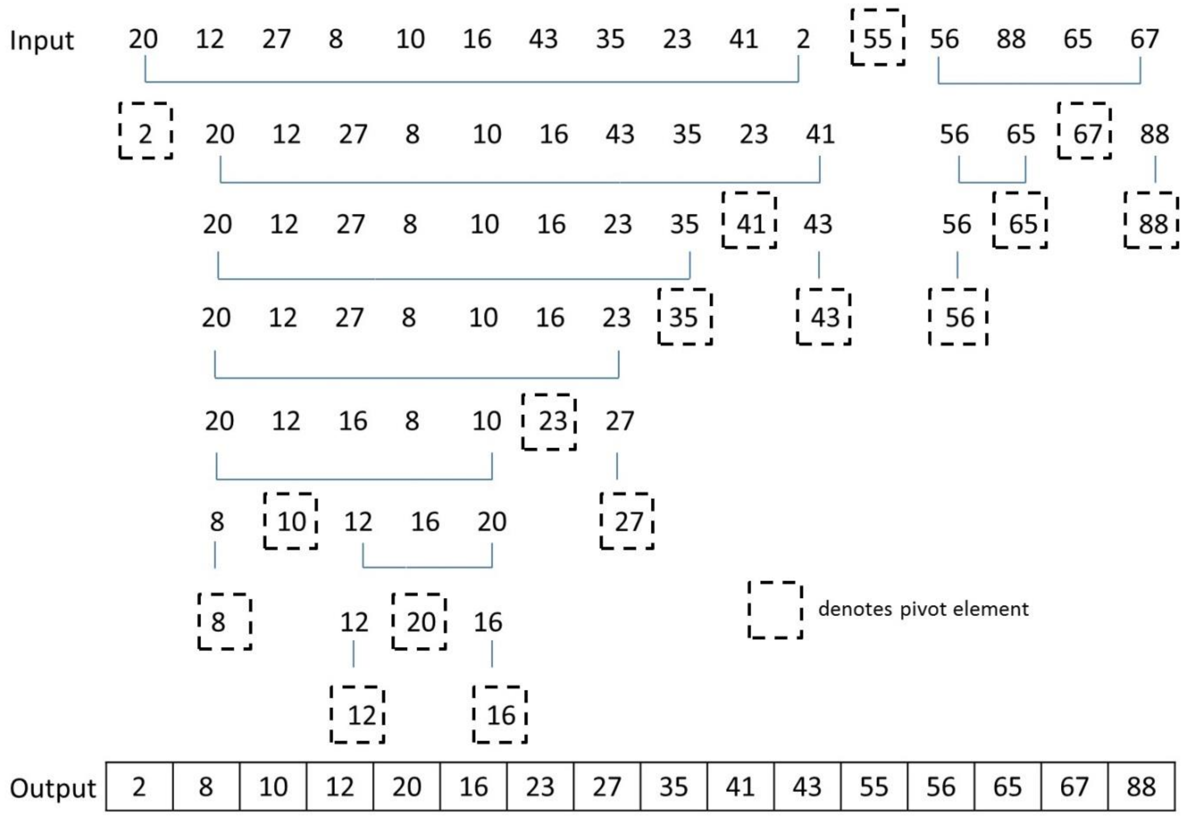

20 12 65 8 10 16 43 35 23 88 2 56 41 27 67 55 Pivot
left
right

Swap 1

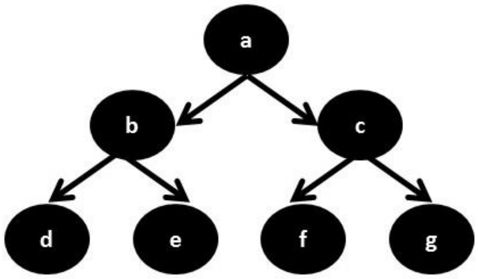

20 12 27 8 10 16 43 35 23 88 2 56 41 65 67 55
left
right

Swap 2

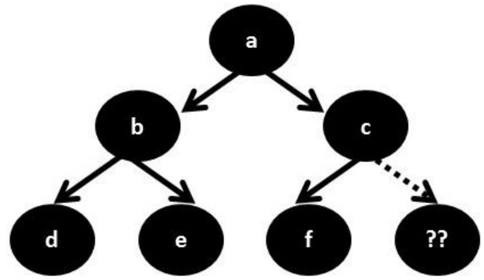

20 12 27 8 10 16 43 35 23 41 2 56 88 65 67 55
left
right



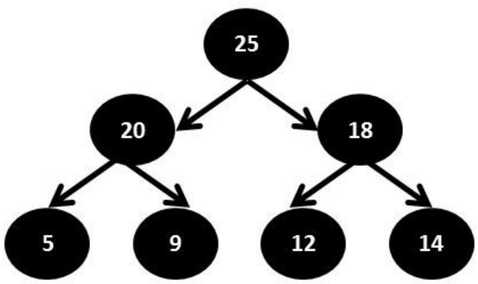
$$\theta(n) = kn + \frac{1}{n}\sum_{s=0}^{n-1}\left[\theta(s) + \theta(n-1-s)\right], \textit{where } k \textit{ is a constant and } \theta(0) = \theta(1) = k$$



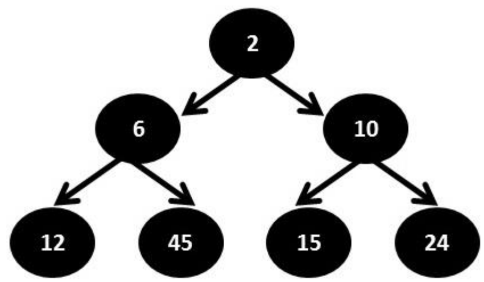
Complete binary tree



In-complete binary tree



Max-heap

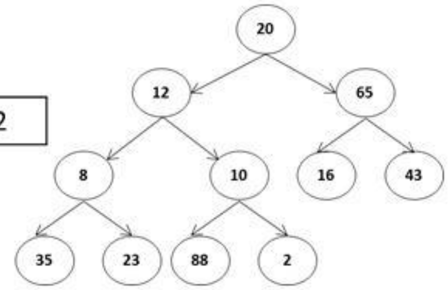


Min-heap

Step 1

20	12	65	8	10	16	43	35	23	88	2

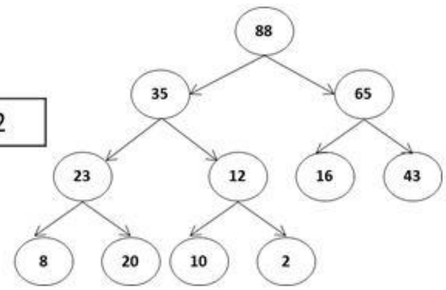
Original input vector (**V**) of length 11



Step 2

88	35	65	23	12	16	43	8	20	10	2

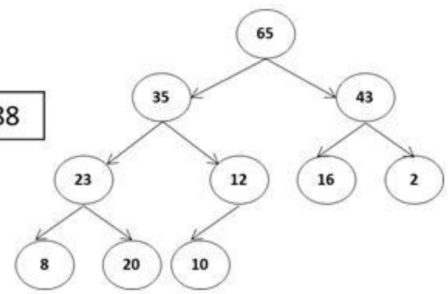
Build initial Max-Heap



Step 3

65	35	43	23	12	16	2	8	20	10	88

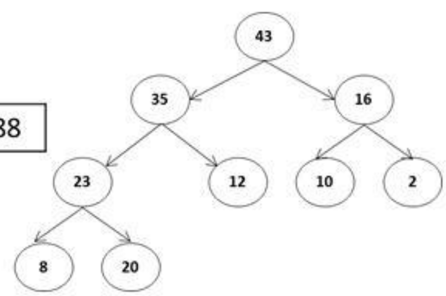
Extract 88 and place in the 11th position



Step 4

43	35	16	23	12	10	2	8	20	65	88

Extract 65 and place in the 10th position



Step n

2	**8**	**10**	**12**	**16**	**20**	**23**	**35**	**43**	**65**	**88**

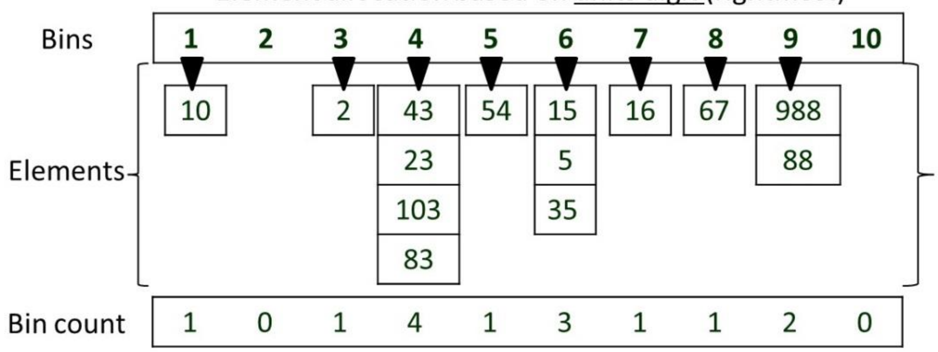
Iteration 0

Input vector

67	**54**	**10**	**988**	**15**	**5**	**16**	**43**	**35**	**23**	**88**	**2**	**103**	**83**

Iteration 1

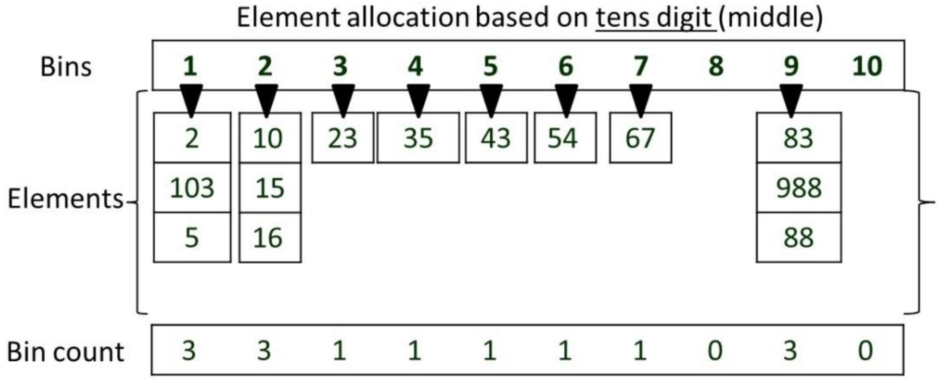
Element allocation based on units digit (rightmost)



Updated vector which is input to iteration 2

10	**2**	**43**	**23**	**103**	**83**	**54**	**15**	**5**	**35**	**16**	**67**	**988**	**88**

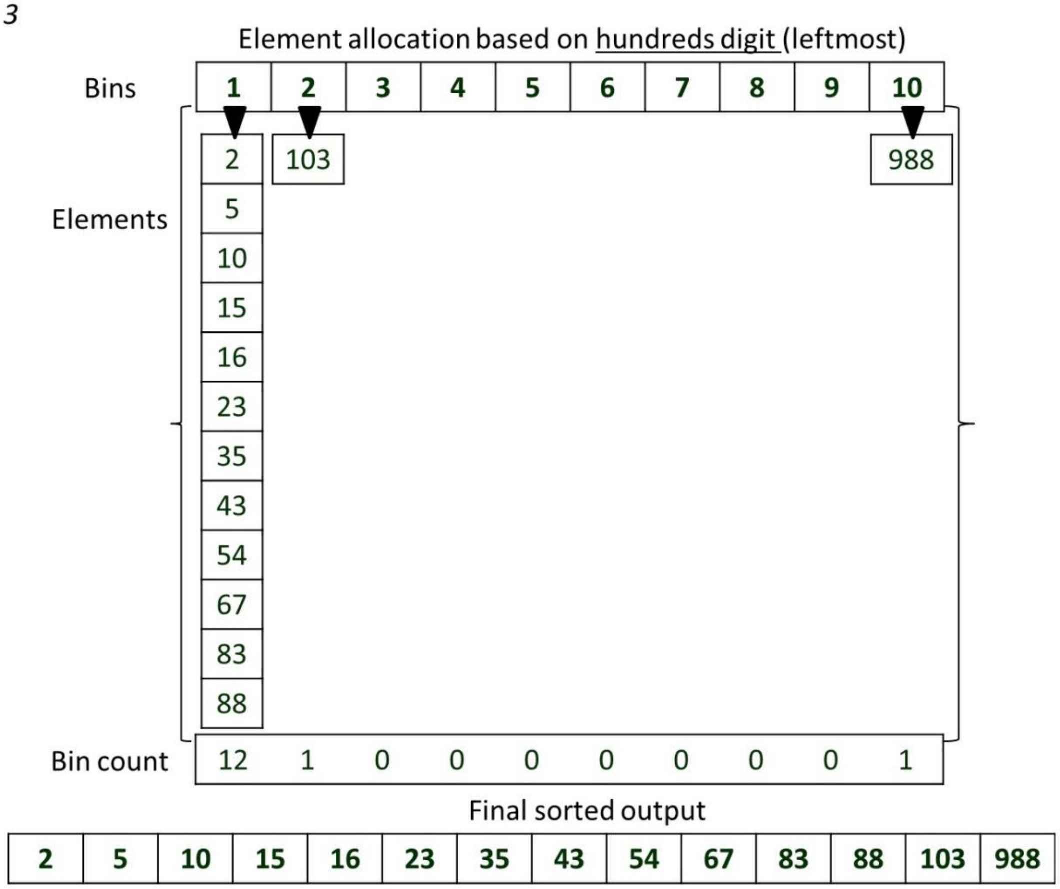
Iteration 2



Updated vector which is input to iteration 3

2	**103**	**5**	**10**	**15**	**16**	**23**	**35**	**43**	**54**	**67**	**83**	**988**	**88**

Iteration 3

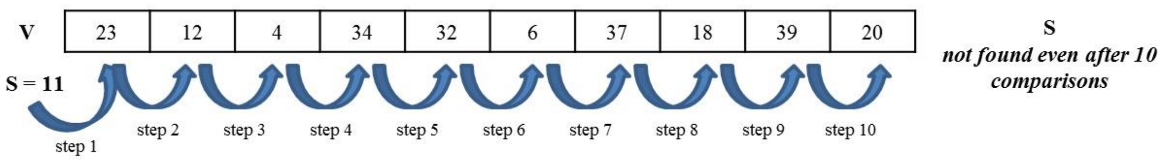


Algorithm	10	100	1k	10k	Best case	Worst case
Insertion sort	0.0818	6.831	757.851	77713.30	2.351	1615.53
Bubble sort	0.0866	14.440	1382.405	140627.75	0.772	2224.58
Selection sort	0.0690	6.453	507.285	46800.13	493.901	479.04
Shell sort	0.0914	1.864	28.038	446.30	14.264	33.47
Merge sort	0.0964	2.836	34.649	491.92	16.687	20.06
Quick sort	0.1115	2.211	26.759	907.60	96.938	691.21
Heap sort	0.1986	4.872	67.710	1887.41	70.570	72.84
Bin sort	0.1658	1.592	31.607	1585.52	28.659	28.42
Radix sort	0.4119	3.206	16.881	276.77	16.948	16.725

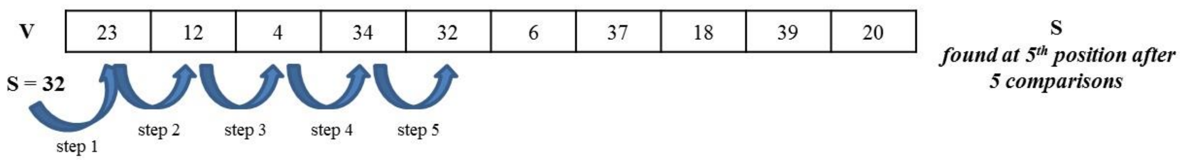
Algorithm	Type of sort	Best case	Average case	Worst case
Insertion sort	Comparison sort	O(n)	$O(n^2)$	$O(n^2)$
Bubble sort	Comparison sort	O(n)	$O(n^2)$	$O(n^2)$
Selection sort	Comparison sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Shell sort	Comparison sort	O(nlog n)	$O(n^{4/3})$	$O(n\log^2 n)$
Merge sort	Comparison sort	O(nlog n)	O(nlog n)	O(nlog n)
Quick sort	Comparison sort	O(nlog n)	O(nlog n)	$O(n^2)$
Heap sort	Comparison sort	O(nlog n)	O(nlog n)	O(nlog n)
Bin sort	Non-comparison sort	-	O(n)	$O(n^2)$
Radix sort	Non-comparison sort	-	O(n)	O(n)

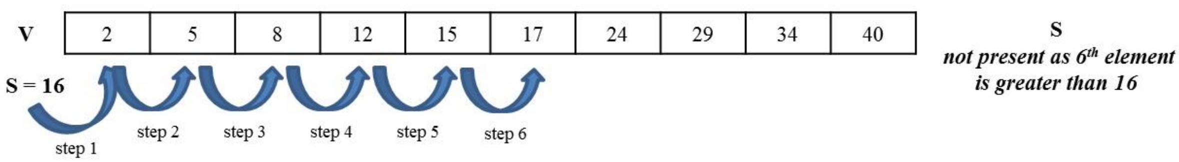
Chapter 6: Exploring Search Options

Part (a)

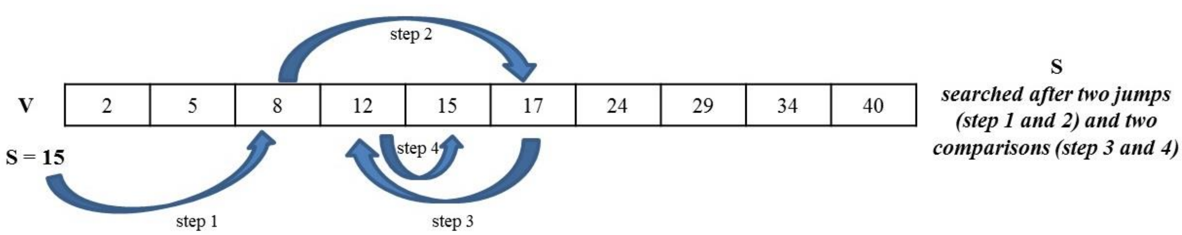


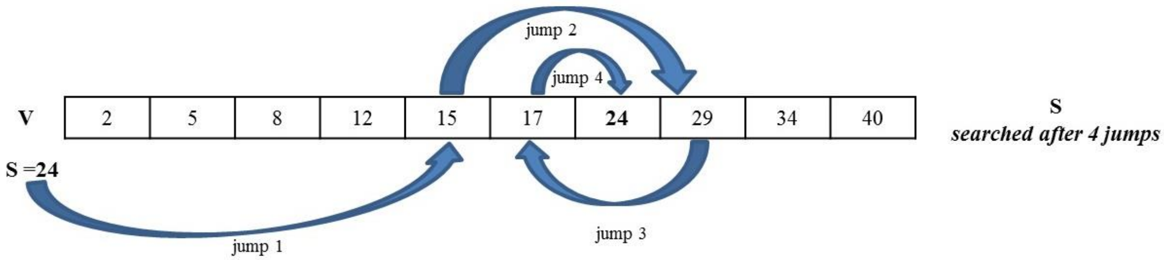
Part (b)



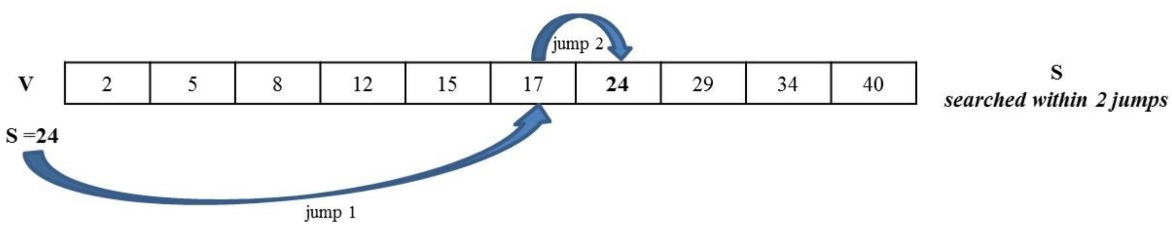


$$\sqrt{n}$$





$$p = \frac{S - V[1]}{V[n] - V[1]}$$



$$C'n = 1p_1 + 2p_2 + \ldots + np_n$$

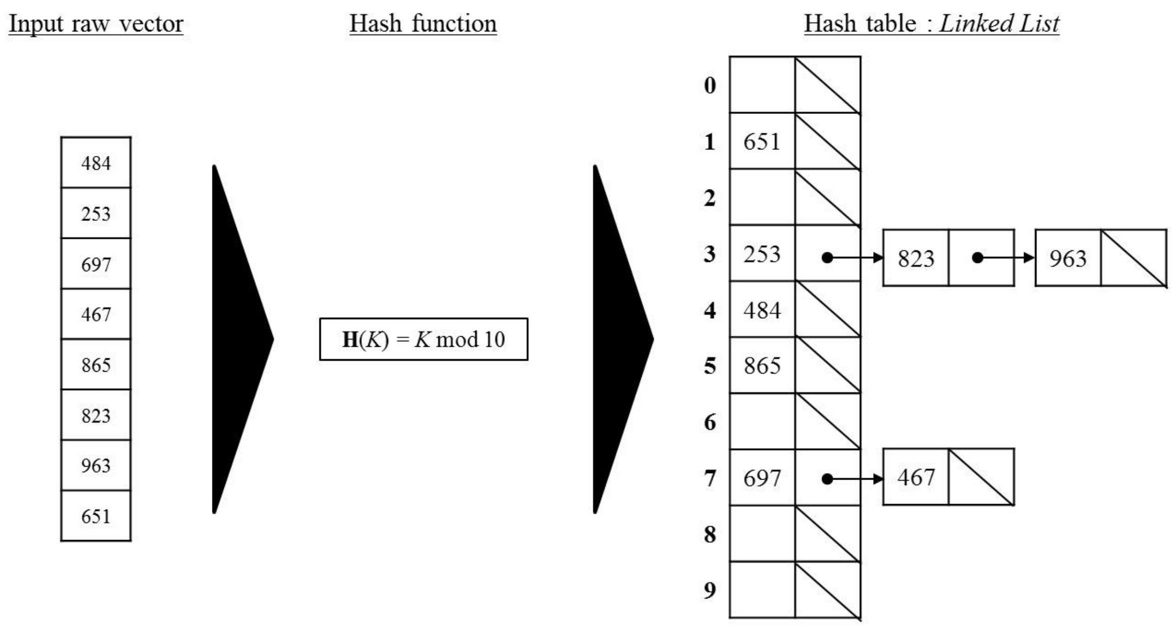
1	2	3	4	5	6	7	8

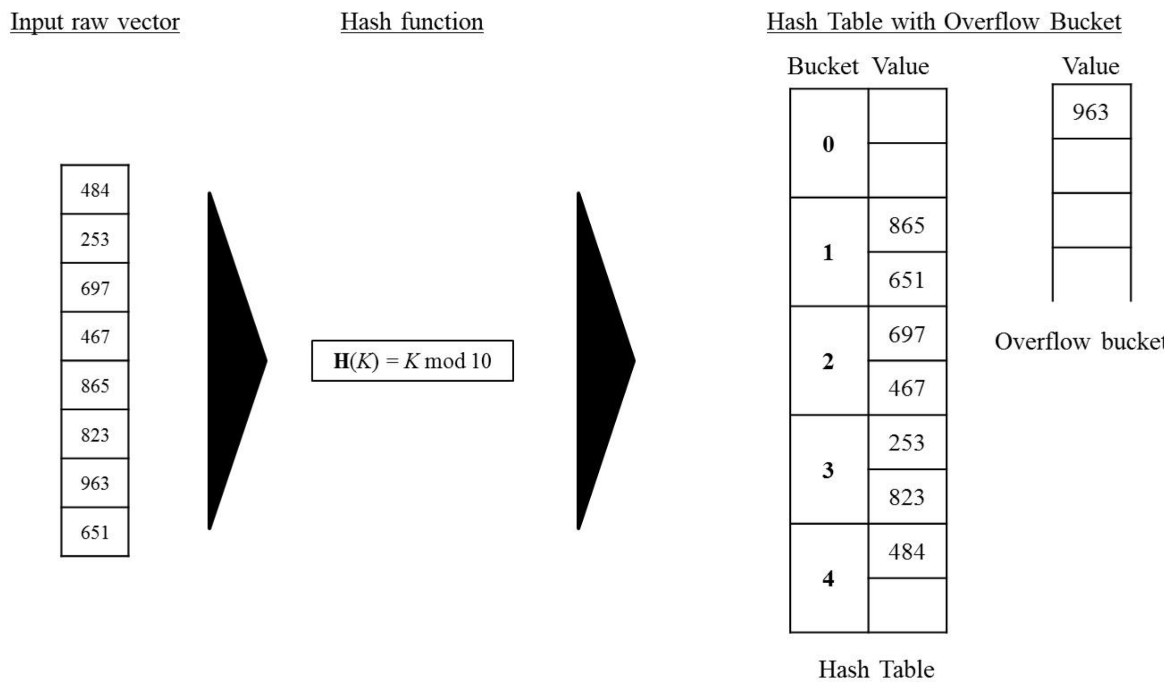
6	4	6	7	5	7	6	1	4	6	7	5

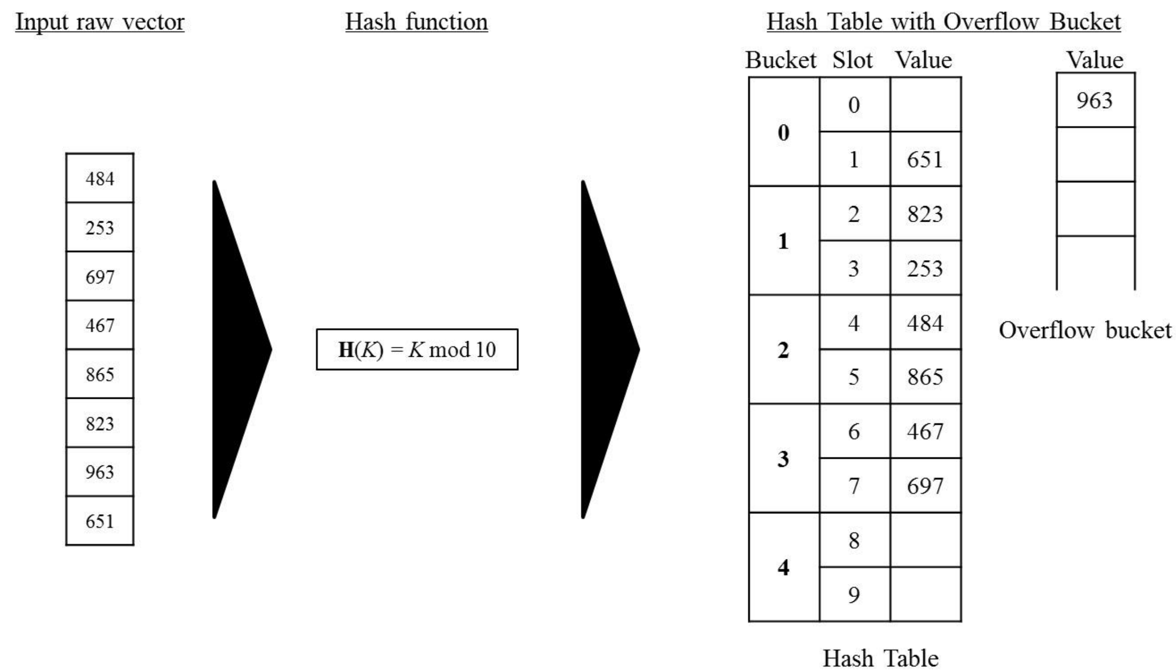
6	7	4	5	1	2	3	8

5	7	6	4	1	2	3	8

1	2	6	4	7	5	3	8




Input raw vector
484
253
697
467
865
823
963
651
Hash function
H(K) = K mod 10
Hash Table with Overflow Bucket
Bucket
Value
0
1
865
651
2
697
467
3
253
823
4
484
Hash Table
Value
963
Overflow bucket



Slot No.	0	1	2	3	4	5	6	7	8	9
Chances	1/10	1/10	0	0	3/10	1/10	0	0	0	4/10

Step 0

Slot No.	Key Values	Probability
0		1/10
1		1/10
2		1/10
3		1/10
4		1/10
5		1/10
6		1/10
7		1/10
8		1/10
9		1/10

Step 1

Slot No.	Key Values	Probability
0		1/10
1		1/10
2		1/10
3	453	0
4		2/10
5		1/10
6		1/10
7		1/10
8		1/10
9		1/10

Step 2

Slot No.	Key Values	Probability
0		1/10
1		1/10
2	362	0
3	453	0
4		3/10
5		1/10
6		1/10
7		1/10
8		1/10
9		1/10

Step 3

Slot No.	Key Values	Probability
0		1/10
1		1/10
2	362	0
3	453	0
4		3/10
5		1/10
6	396	0
7		2/10
8		1/10
9		1/10

Step 4

Slot No.	Key Values	Probability
0		1/10
1		1/10
2	362	0
3	453	0
4		3/10
5		1/10
6	396	0
7	156	0
8		3/10
9		1/10

Step5

Slot No.	Key Values	Probability
0		1/10
1		1/10
2	362	0
3	453	0
4		3/10
5		1/10
6	396	0
7	156	0
8	957	0
9		4/10

$$\alpha = \frac{N}{M}$$

$$\frac{N(N-1)\ldots(N-i-1)}{M(M-1)\ldots(M-i-1)}$$

$$1 + \sum_{i=1}^{\infty} (N/M)^i \approx \frac{1}{1-\alpha}$$

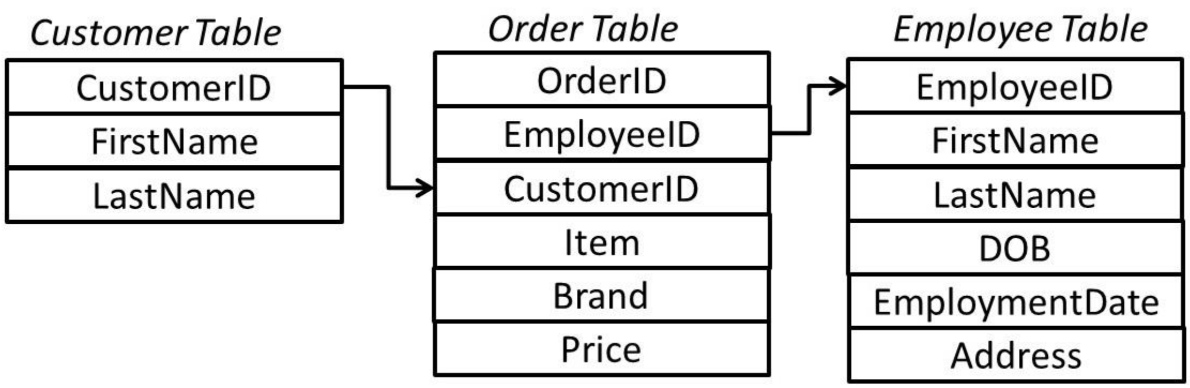
$$\frac{1}{\alpha}\int_0^{\alpha} \frac{1}{1-x}\,dx = \frac{1}{\alpha} log_e \frac{1}{1-\alpha}$$

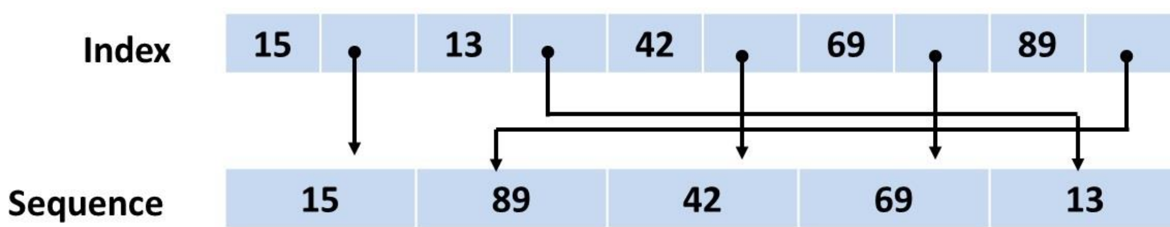
$$\frac{1}{2}\left(1 + \frac{1}{(1+\alpha)^2}\right)$$

$$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)}\right)$$

$$\sqrt{n}$$

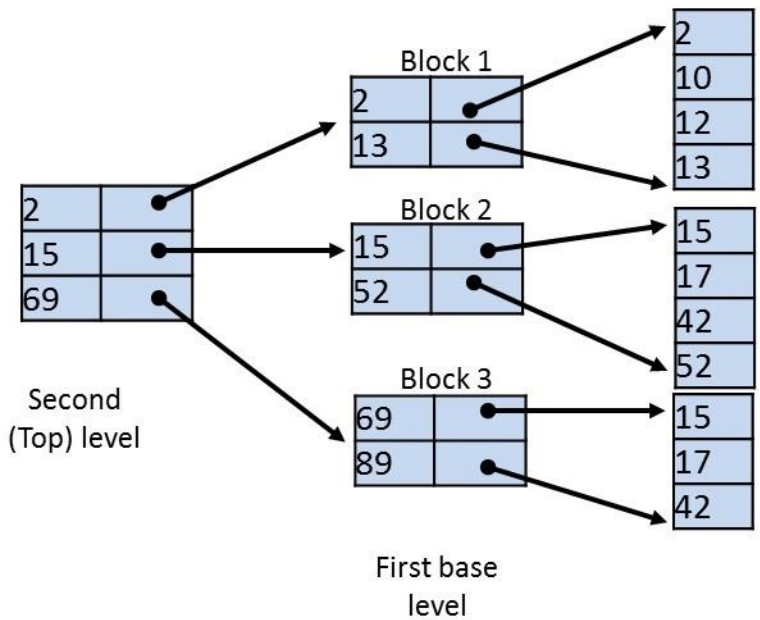
Chapter 7: Indexing



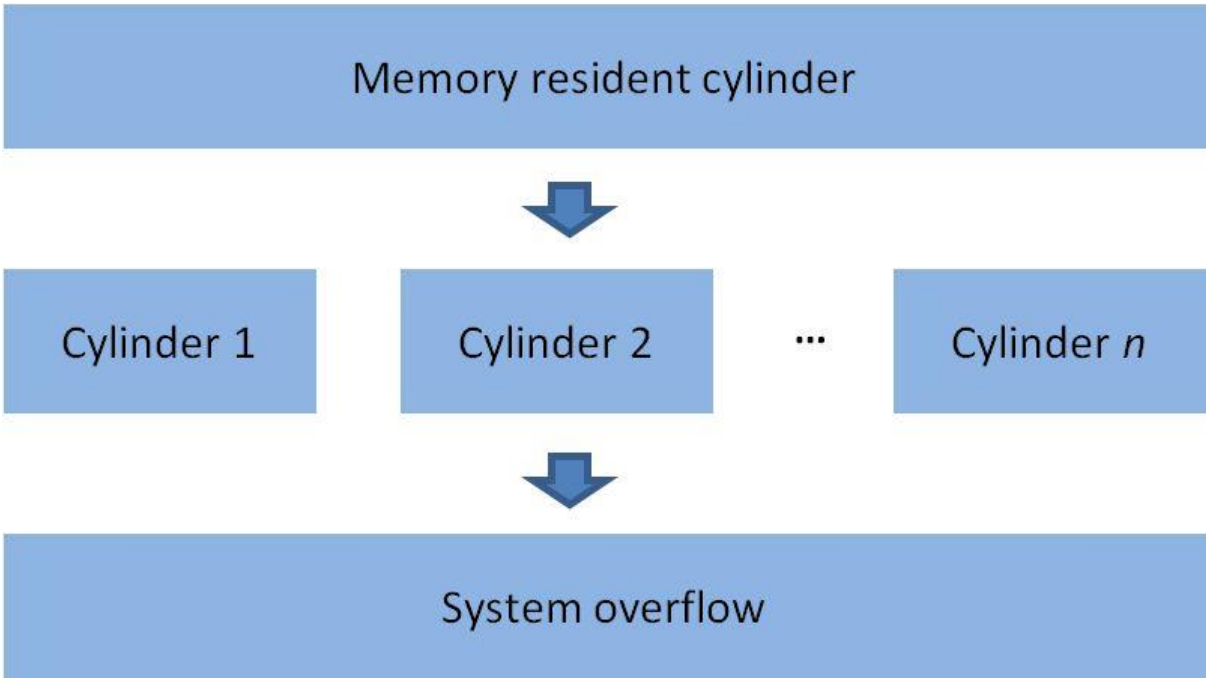


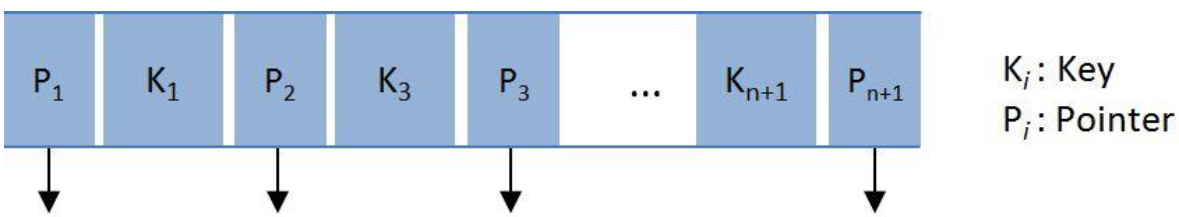
15	**52**	**13**	**12**	**42**	**17**	**69**	**10**	**89**	**75**	**2**

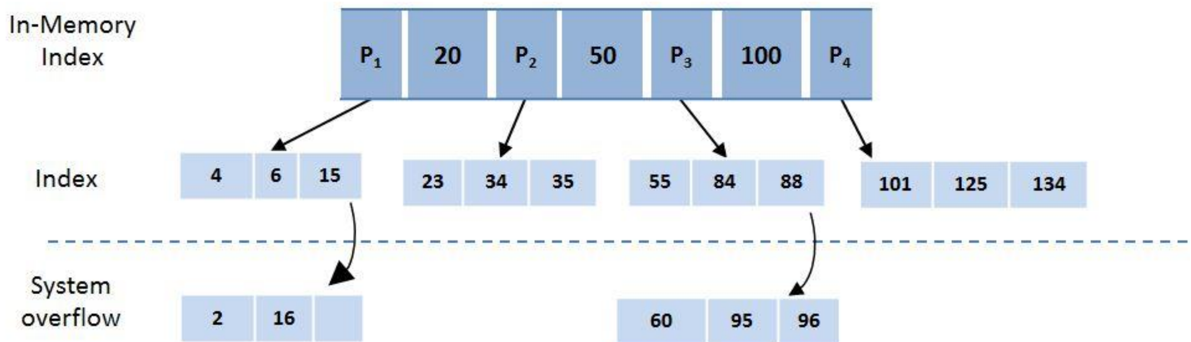
a) Data array



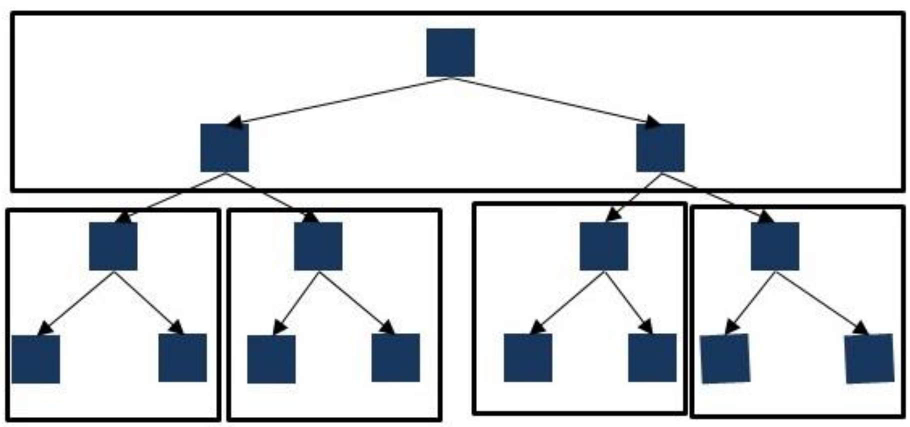
b) Example of second level index

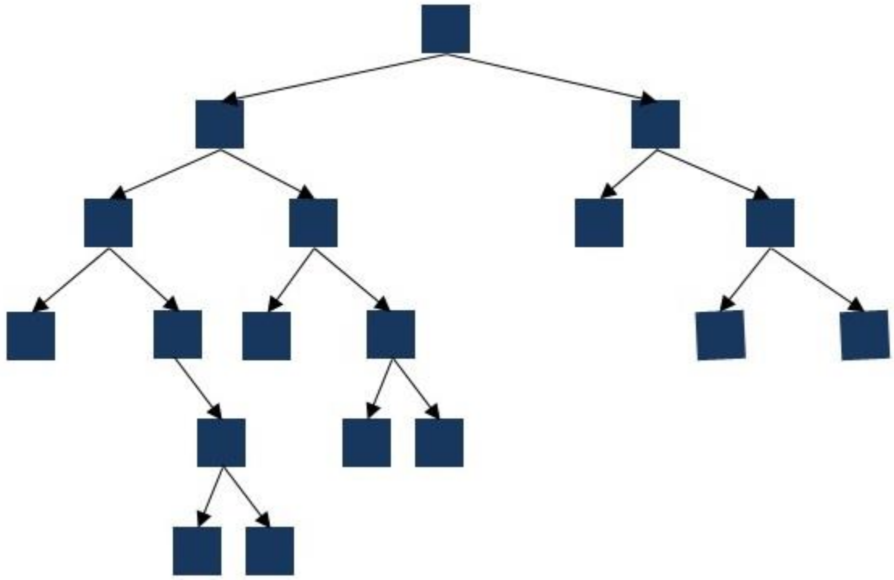

Memory resident cylinder
Cylinder 1
Cylinder 2
...
Cylinder *n*
System overflow

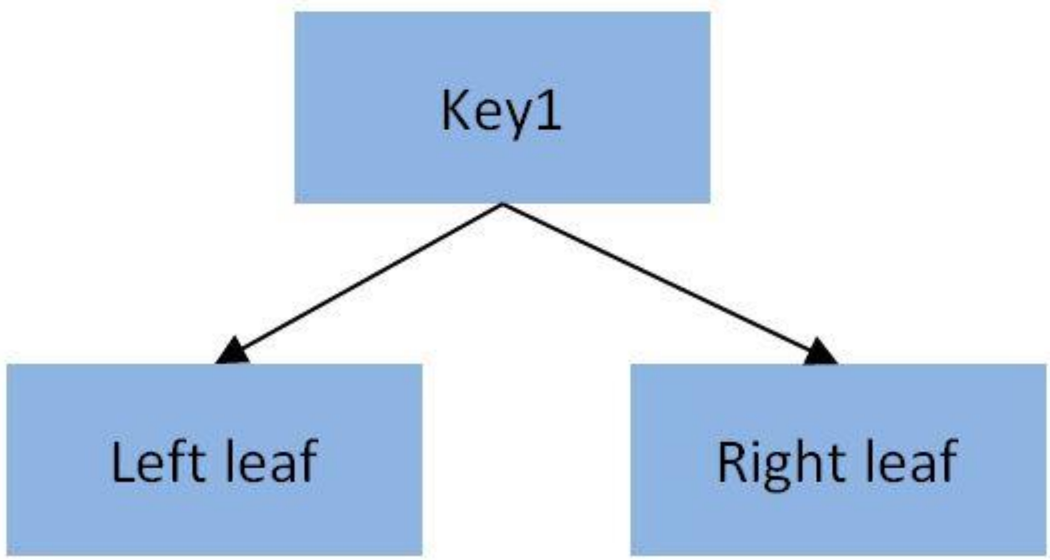

P_1
K_1
P_2
K_3
P_3
...
K_{n+1}
P_{n+1}
K_i : Key
P_i : Pointer

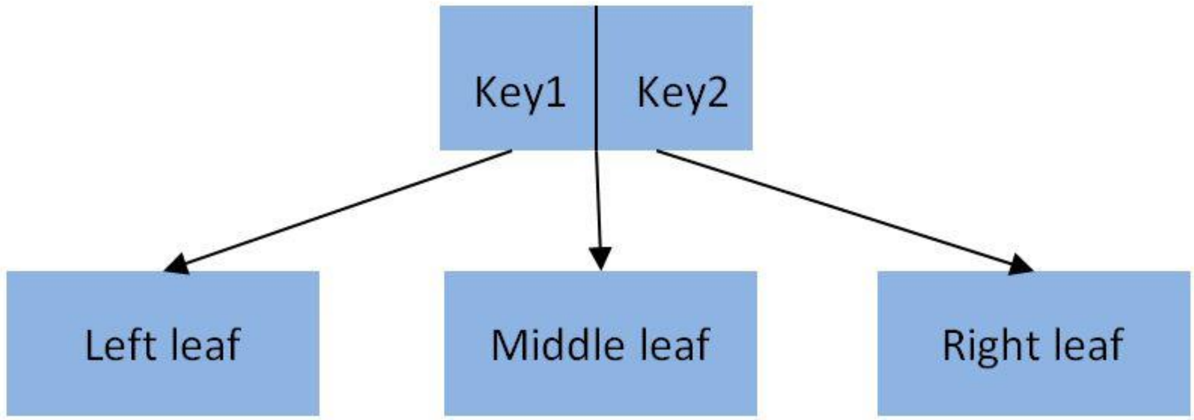


In-Memory Index
P1
20
P2
50
P3
100
P4
Index
4
6
15
23
34
35
55
84
88
101
125
134
System overflow
2
16
60
95
96

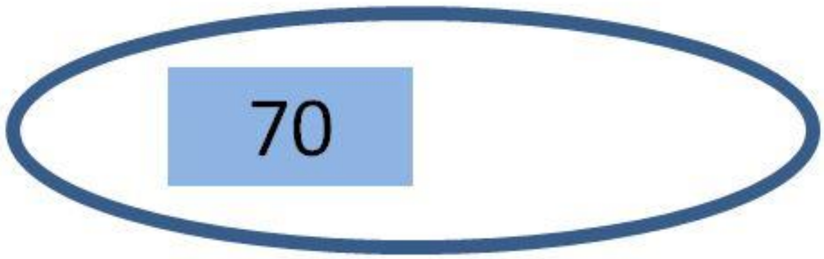




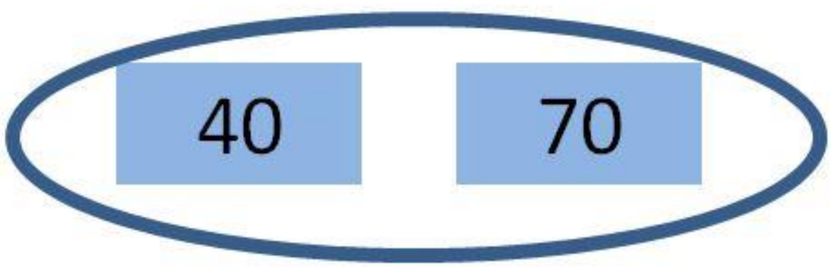

Key1
Left leaf
Right leaf



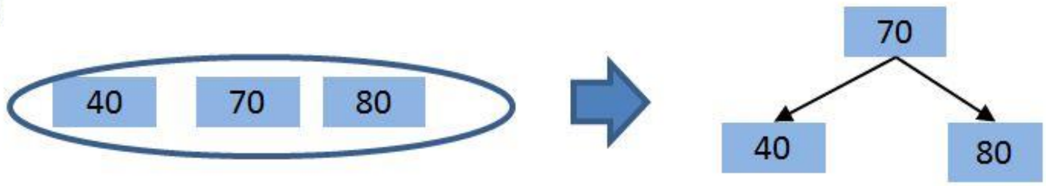
Insert 70



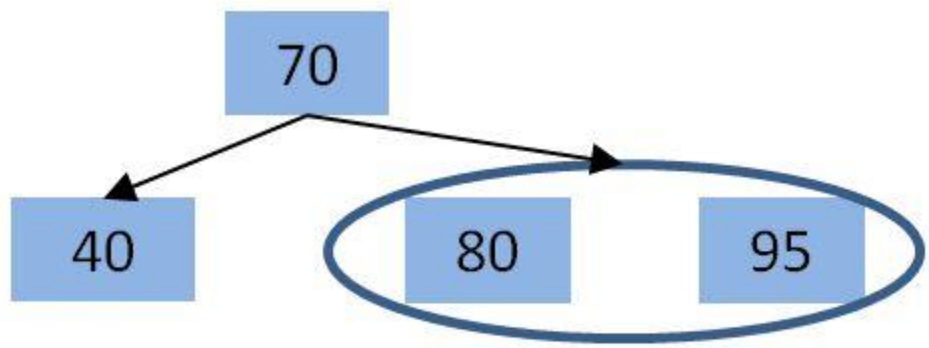
Insert 40



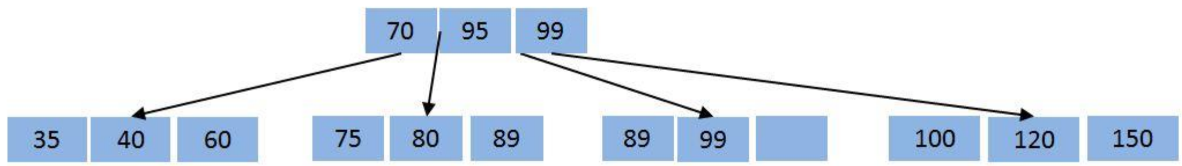
Insert 80

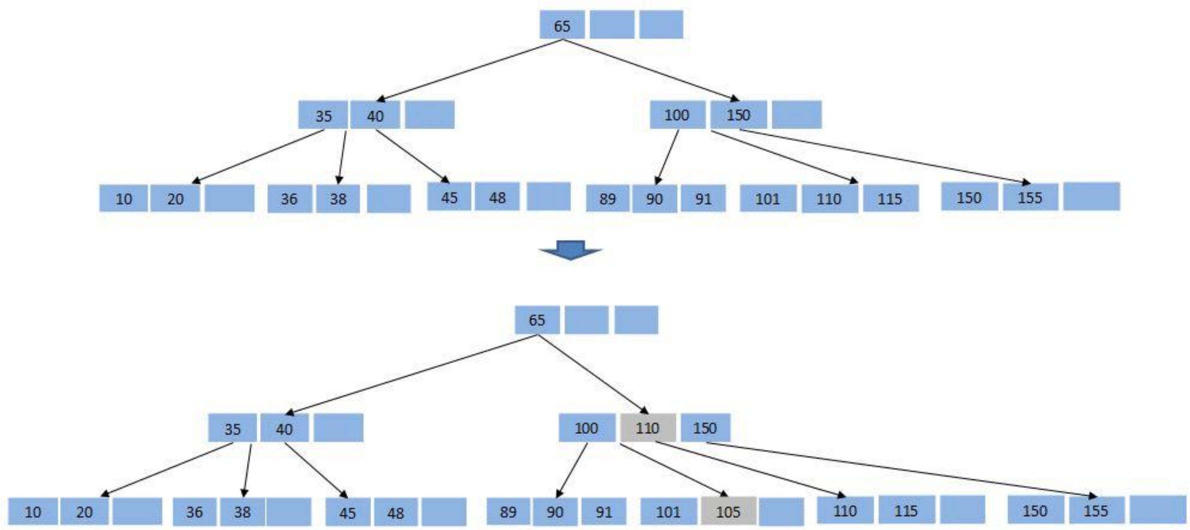


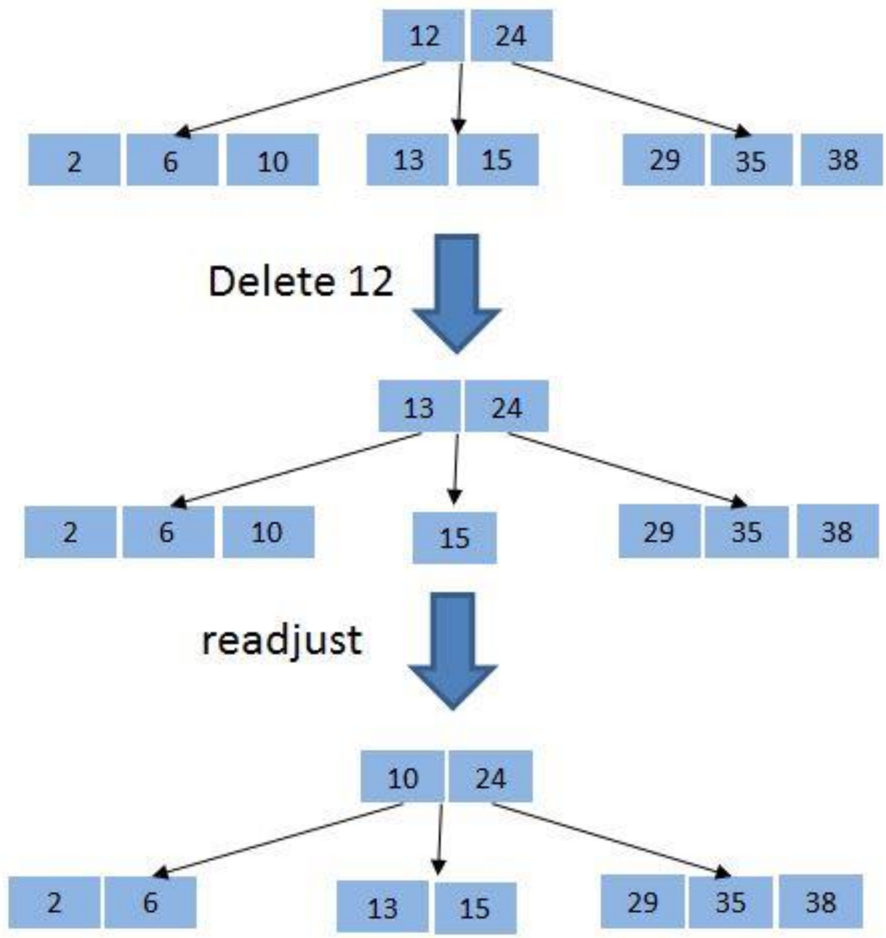
Insert 95

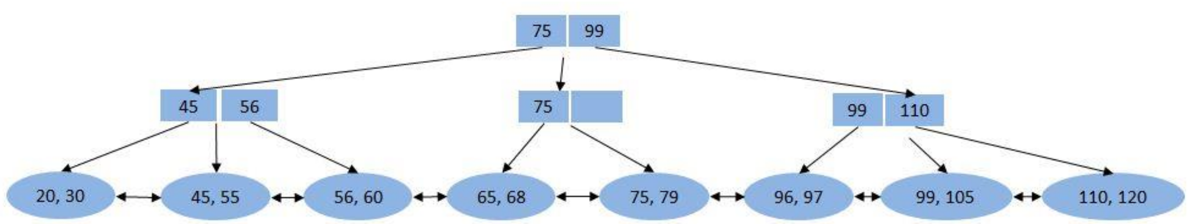


Insert 99

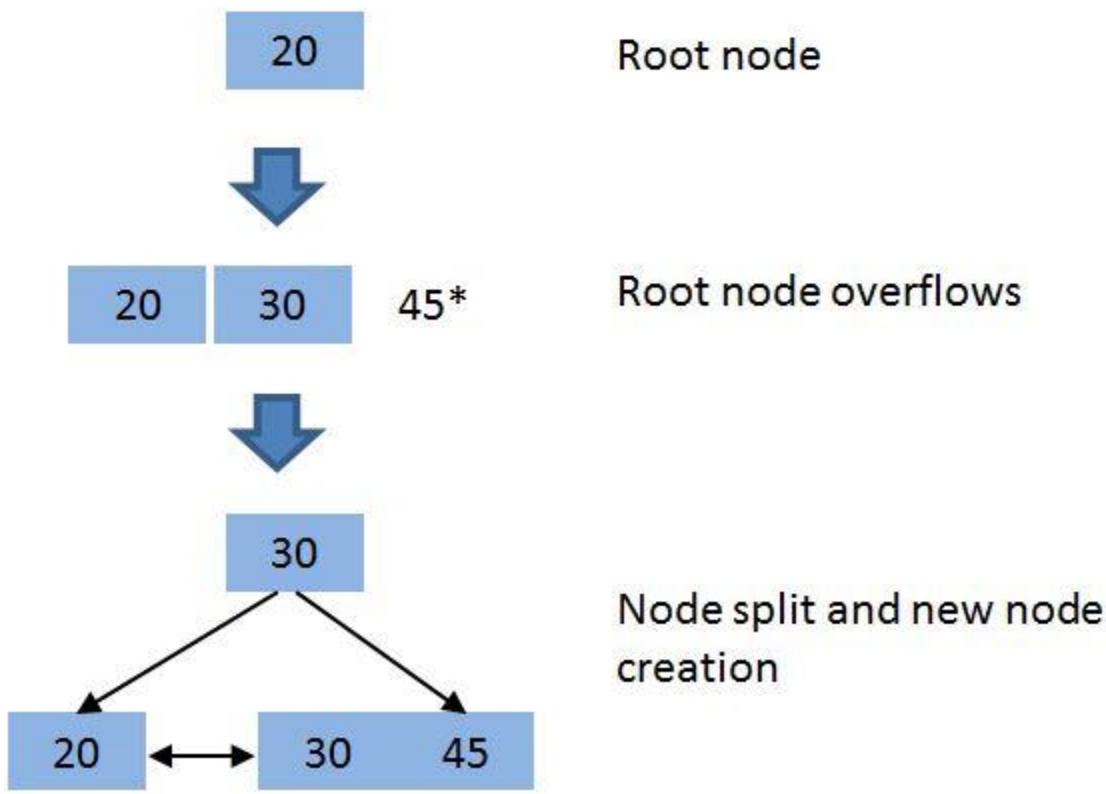

70 95 99
35 40 60
75 80 89
89 99
100 120 150

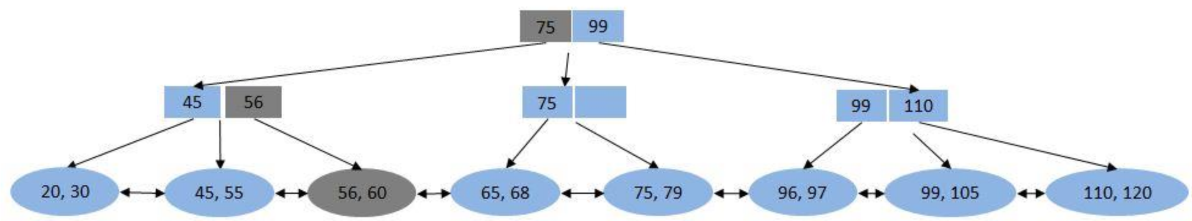

65
35 40
100 150
10 20
36 38
45 48
89 90 91
101 110 115
150 155
65
35 40
100 110 150
10 20
36 38
45 48
89 90 91
101 105
110 115
150 155

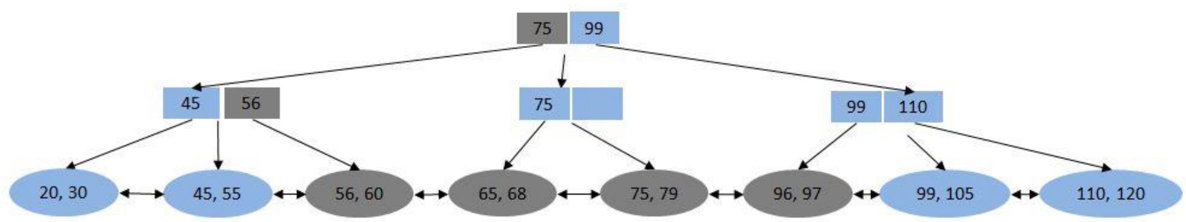

12
24
2
6
10
13
15
29
35
38
Delete 12
13
24
2
6
10
15
29
35
38
readjust
10
24
2
6
13
15
29
35
38

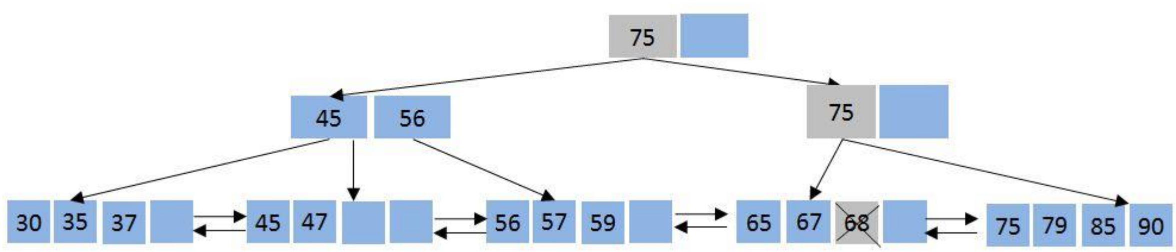

75
99
45
56
75
99
110
20, 30
45, 55
56, 60
65, 68
75, 79
96, 97
99, 105
110, 120

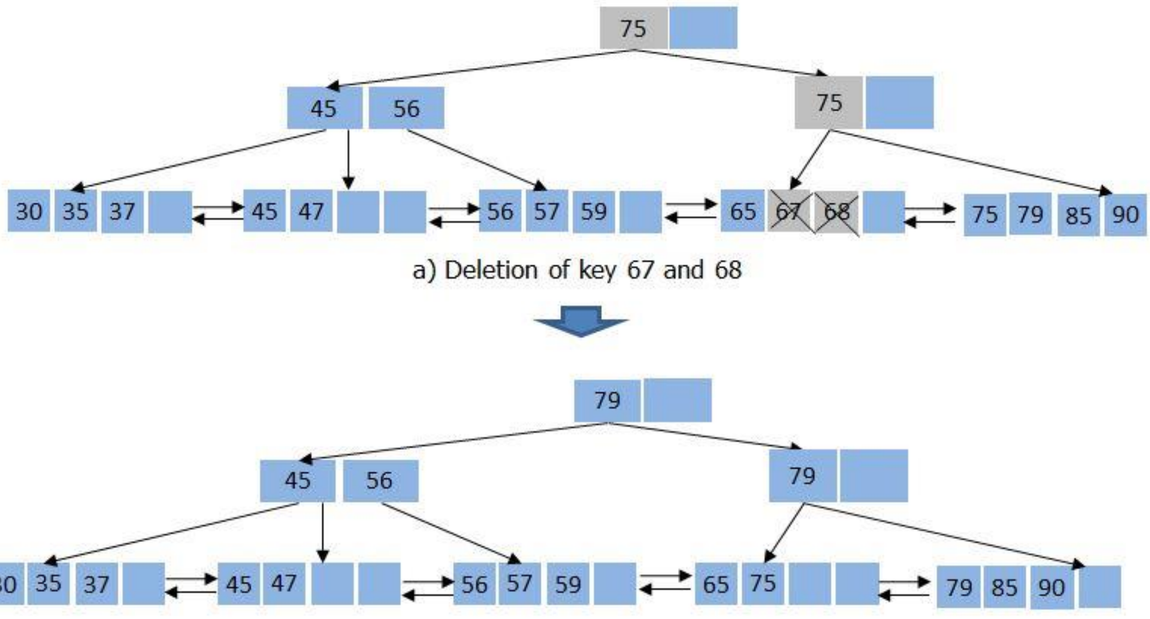
```
> dlinkchildNode(1)
<environment: 0x000000000d034c88>
attr(,"class")
[1] " dlinkchildNode"
```




75
99
45
56
75
99
110
20, 30
45, 55
56, 60
65, 68
75, 79
96, 97
99, 105
110, 120


75
99
45
56
75
99
110
20, 30
45, 55
56, 60
65, 68
75, 79
96, 97
99, 105
110, 120

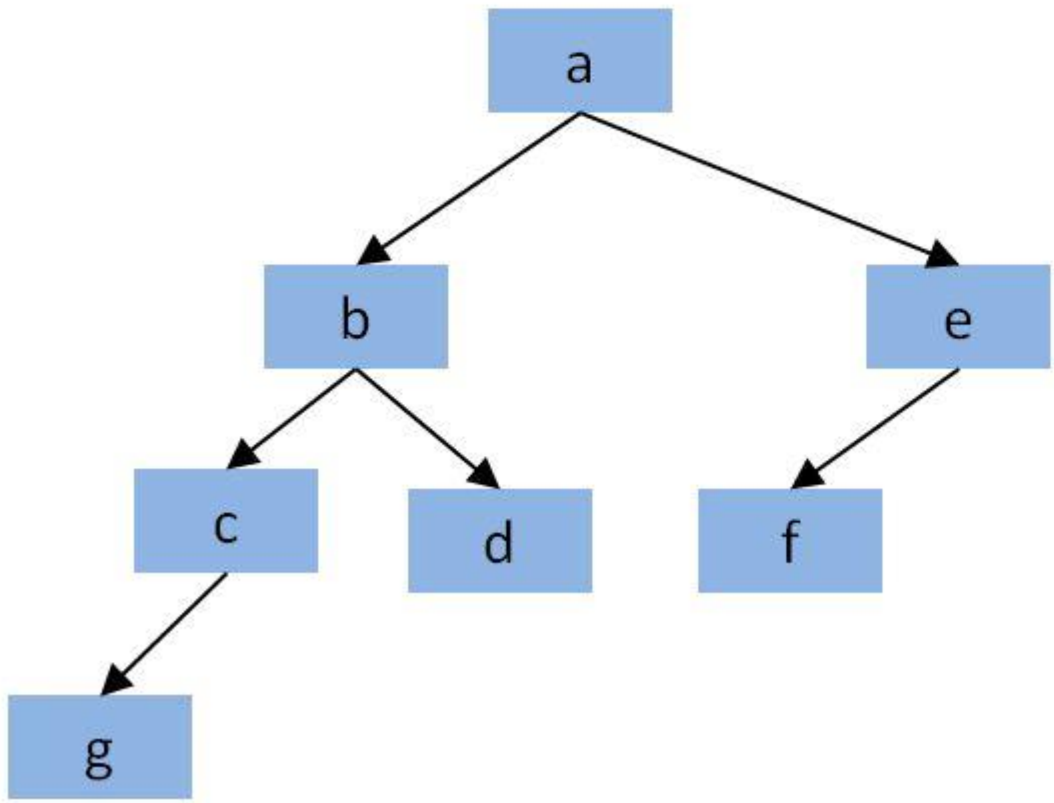

75
45
56
75
30
35
37
45
47
56
57
59
65
67
68
75
79
85
90



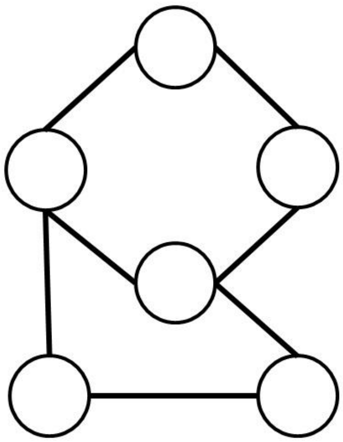
a) Deletion of key 67 and 68

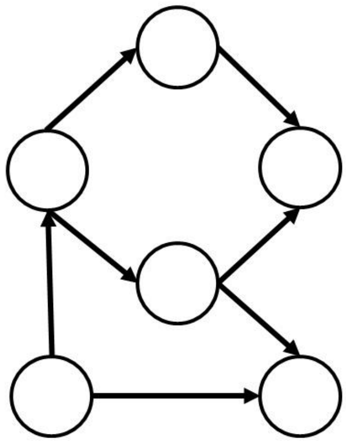
b) B+-tree readjust to balance the tree

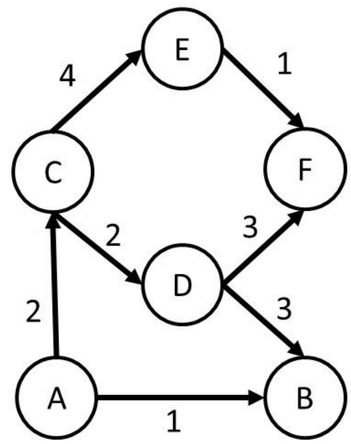
Depth	Average number of object	# of leaf nodes
d=0	133	1
d=1	133^2=17,689	133
d=2	133^3=2,352,637	17,689
d=3	133^4=312,900,721	2,352,637
…	…	…
d=n	133^n	133^{n-1}

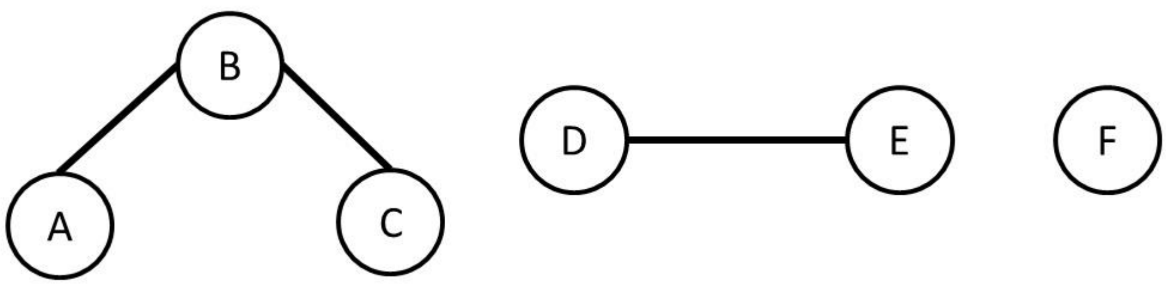

a
b
e
c
d
f
g

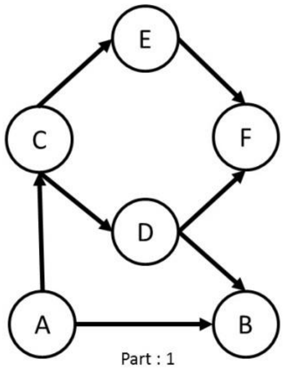
Chapter 8: Graphs



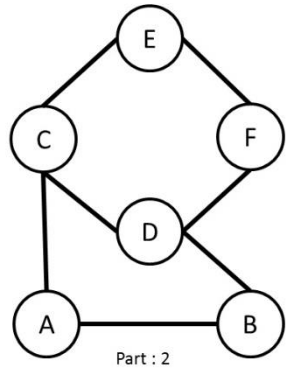








Part : 1



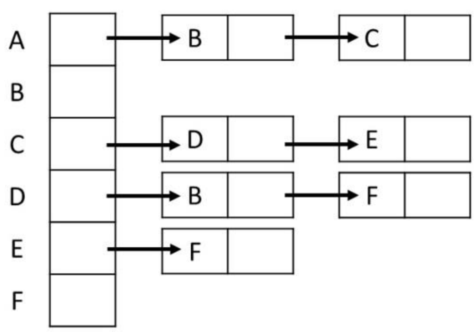
Part : 2

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	0	0	0
C	0	0	0	1	1	0
D	0	1	0	0	0	1
E	0	0	0	0	0	1
F	0	0	0	0	0	0

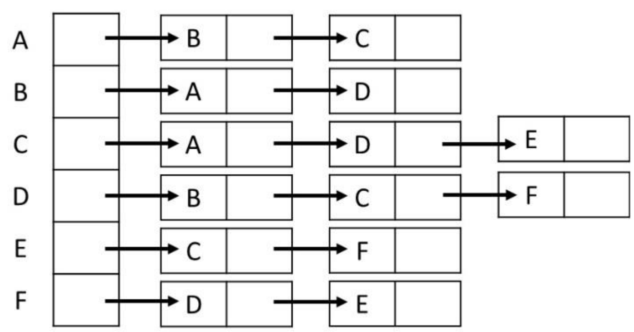
Part : 3

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	0	0	1	0	0
C	1	0	0	1	1	0
D	0	1	1	0	0	1
E	0	0	1	0	0	1
F	0	0	0	1	1	0

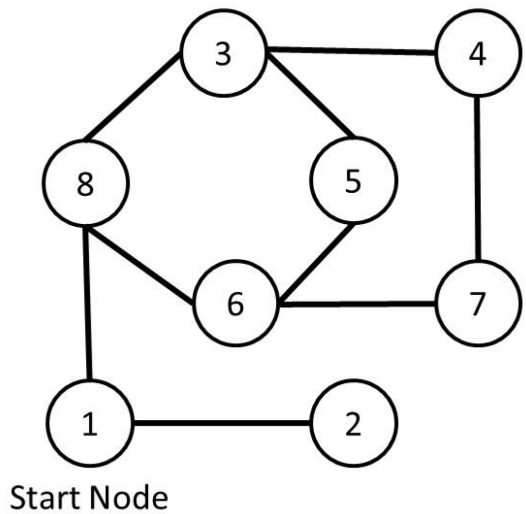
Part : 4

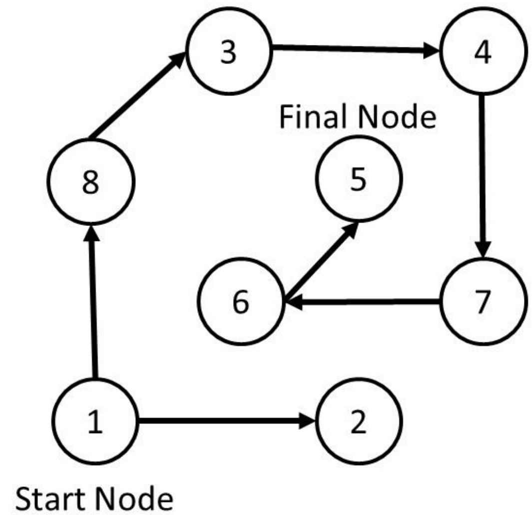


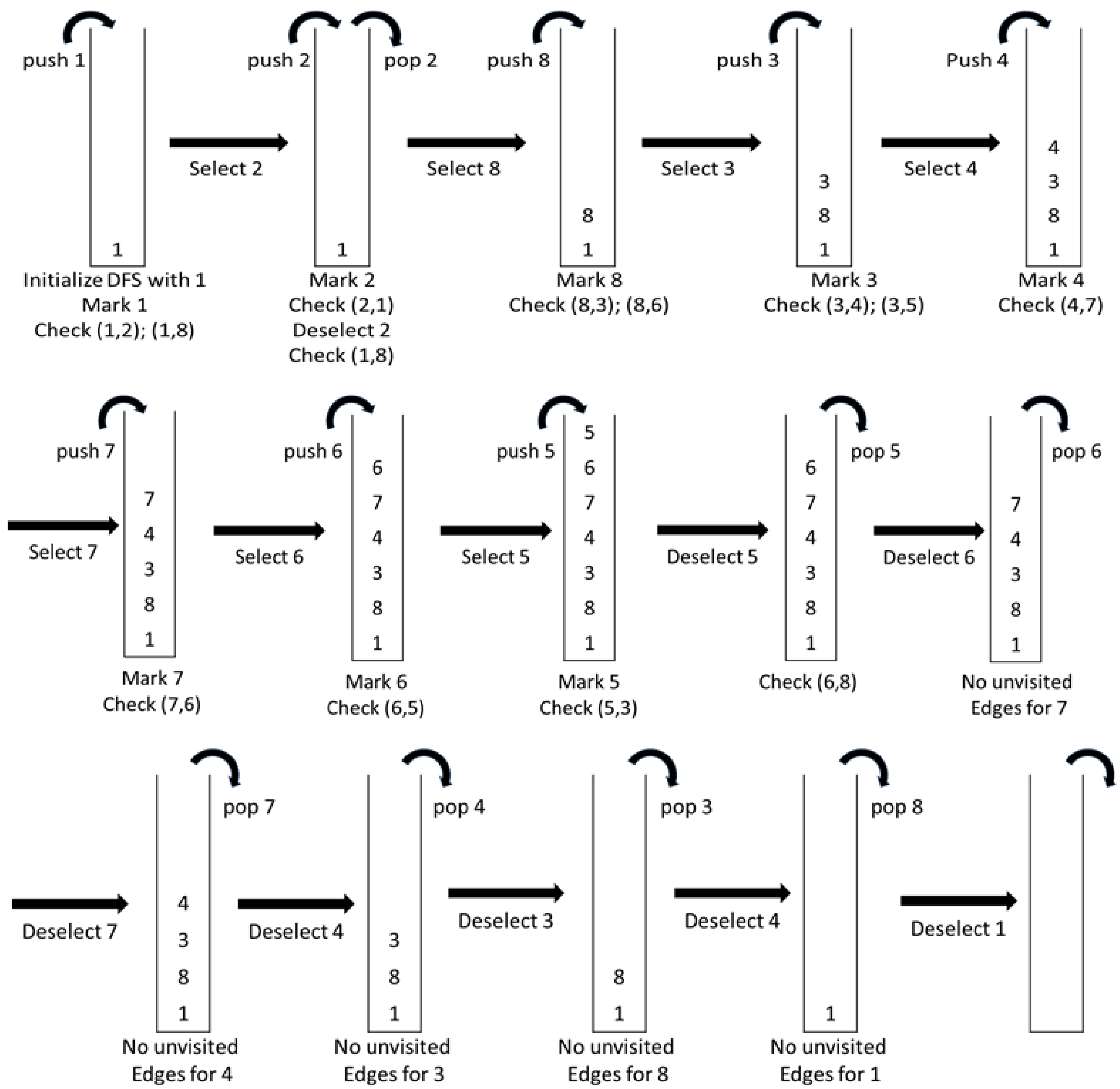
Part : 5



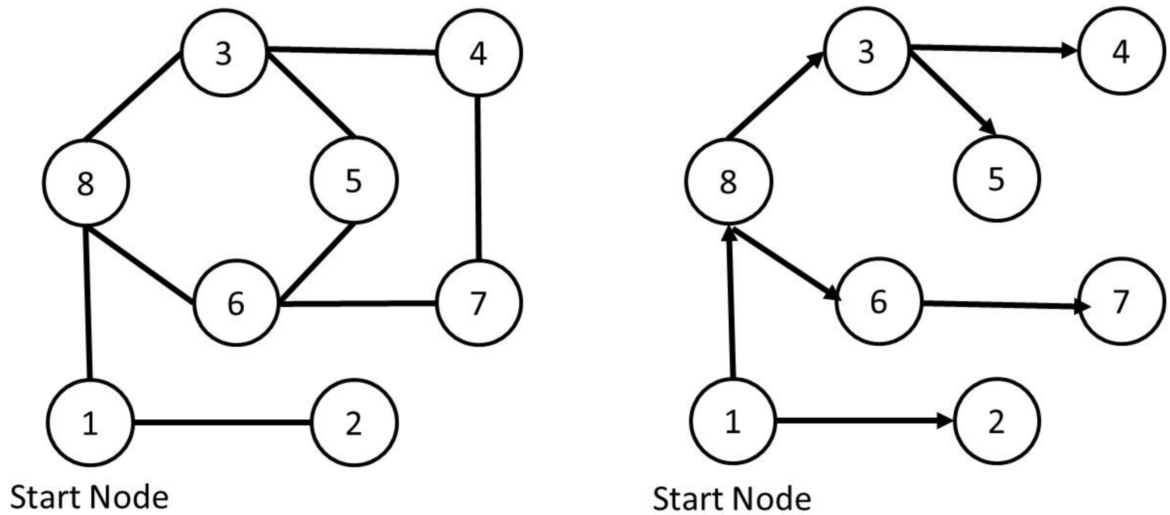
Part : 6

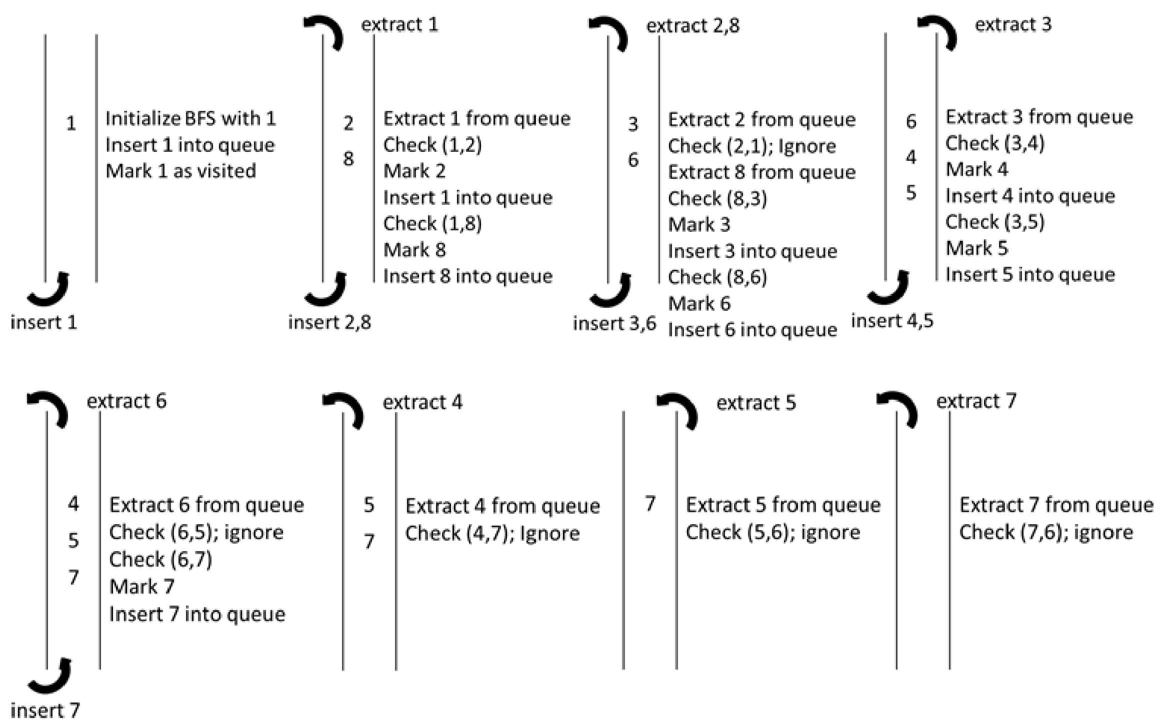

3
4
8
5
6
7
1
2
Start Node

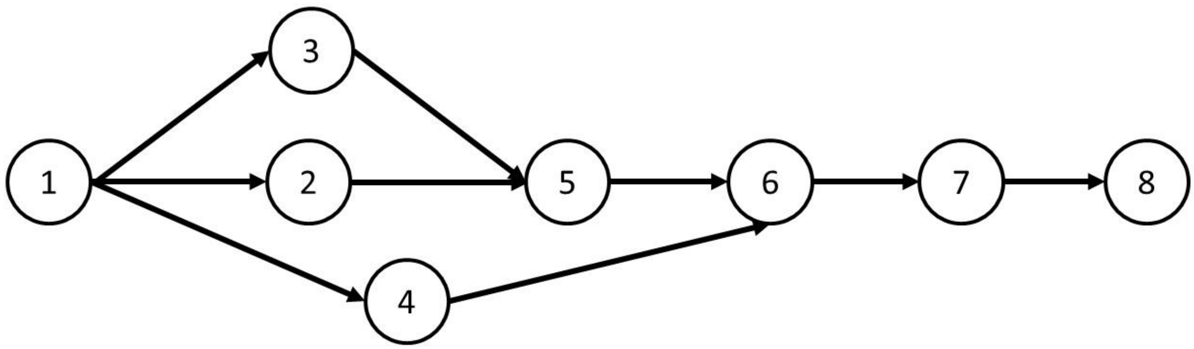

3
4
Final Node
8
5
6
7
1
2
Start Node

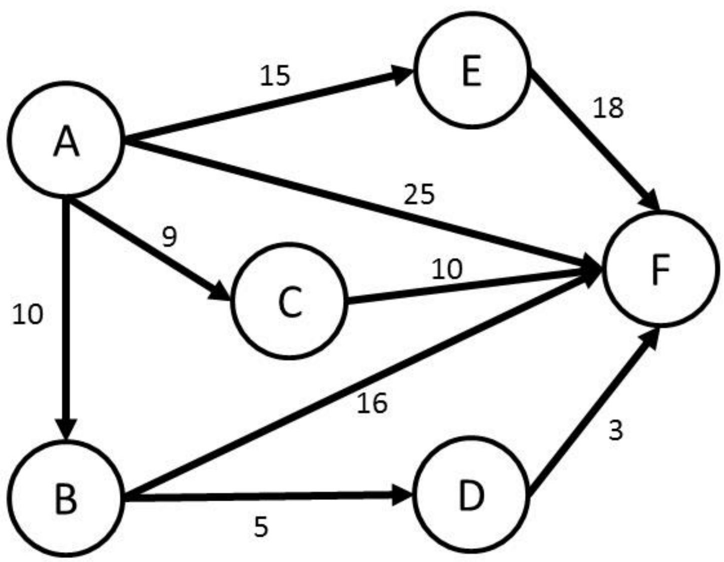


push 1
1
Initialize DFS with 1
Mark 1
Check (1,2); (1,8)
Select 2
push 2
pop 2
1
Mark 2
Check (2,1)
Deselect 2
Check (1,8)
Select 8
push 8
8
1
Mark 8
Check (8,3); (8,6)
Select 3
push 3
3
8
1
Mark 3
Check (3,4); (3,5)
Select 4
Push 4
4
3
8
1
Mark 4
Check (4,7)
Select 7
push 7
7
4
3
8
1
Mark 7
Check (7,6)
Select 6
push 6
6
7
4
3
8
1
Mark 6
Check (6,5)
Select 5
push 5
5
6
7
4
3
8
1
Mark 5
Check (5,3)
Deselect 5
pop 5
6
7
4
3
8
1
Check (6,8)
Deselect 6
pop 6
7
4
3
8
1
No unvisited
Edges for 7
Deselect 7
pop 7
4
3
8
1
No unvisited
Edges for 4
Deselect 4
pop 4
3
8
1
No unvisited
Edges for 3
Deselect 3
pop 3
8
1
No unvisited
Edges for 8
Deselect 4
pop 8
1
No unvisited
Edges for 1
Deselect 1

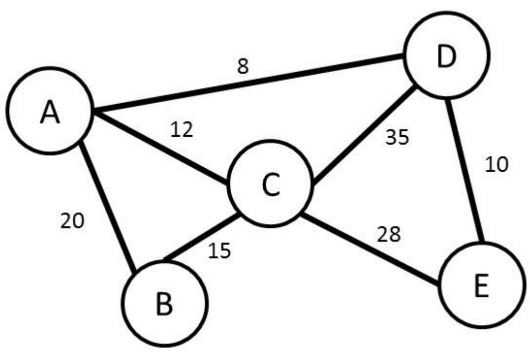

3
4
8
5
6
7
1
2
Start Node
3
4
8
5
6
7
1
2
Start Node

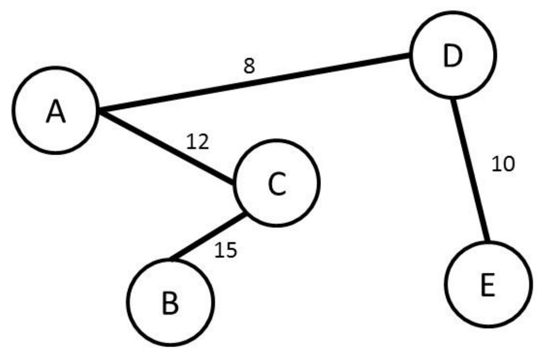

1
Initialize BFS with 1
Insert 1 into queue
Mark 1 as visited
insert 1
extract 1
2
8
Extract 1 from queue
Check (1,2)
Mark 2
Insert 1 into queue
Check (1,8)
Mark 8
Insert 8 into queue
insert 2,8
extract 2,8
3
6
Extract 2 from queue
Check (2,1); Ignore
Extract 8 from queue
Check (8,3)
Mark 3
Insert 3 into queue
Check (8,6)
Mark 6
Insert 6 into queue
insert 3,6
extract 3
6
4
5
Extract 3 from queue
Check (3,4)
Mark 4
Insert 4 into queue
Check (3,5)
Mark 5
Insert 5 into queue
insert 4,5
extract 6
4
5
7
Extract 6 from queue
Check (6,5); ignore
Check (6,7)
Mark 7
Insert 7 into queue
insert 7
extract 4
5
7
Extract 4 from queue
Check (4,7); Ignore
extract 5
7
Extract 5 from queue
Check (5,6); ignore
extract 7
Extract 7 from queue
Check (7,6); ignore

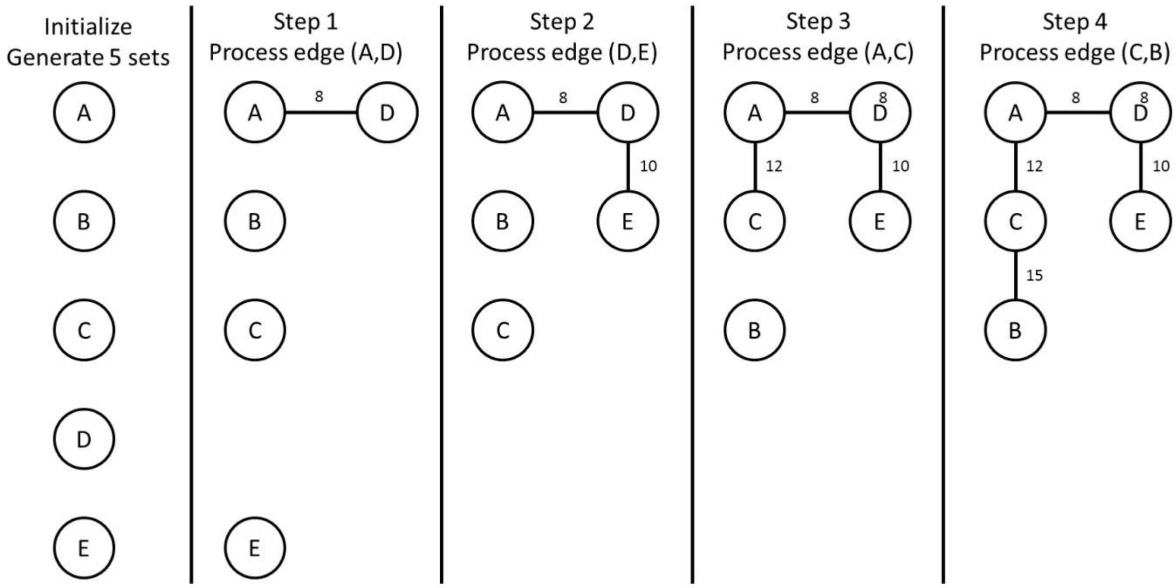

1
2
3
4
5
6
7
8

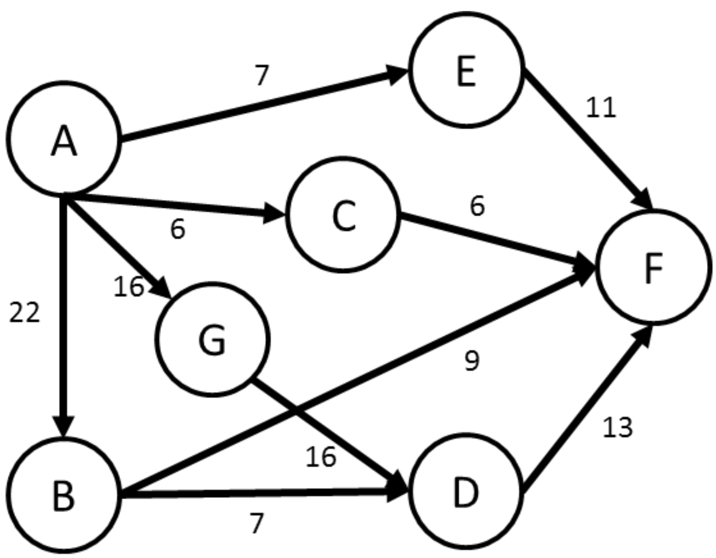

A
B
C
D
E
F
15
18
25
9
10
10
16
3
5

	A	B	C	D	E	F
Initialise	0	∞	∞	∞	∞	∞
Extract A	0	10	9	∞	15	25
Extract C	0	10	9	∞	15	19
Extract B	0	10	9	15	15	19
Extract D	0	10	9	15	15	18
Extract E	0	10	9	15	15	18

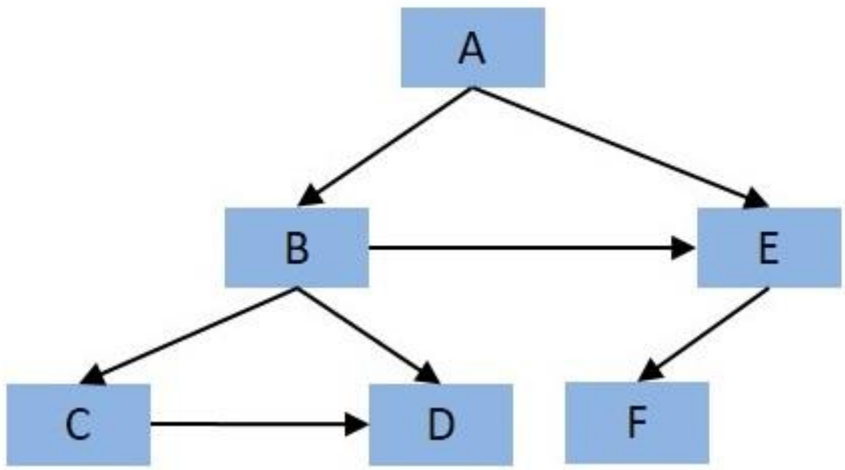


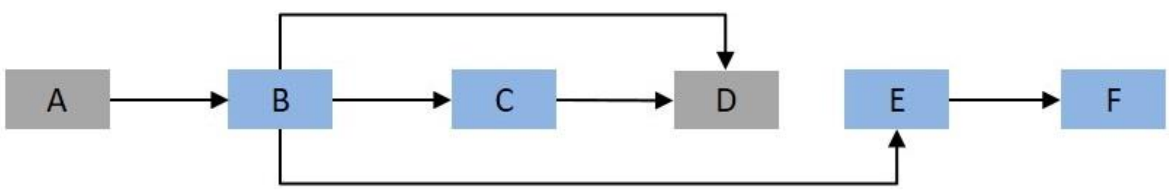


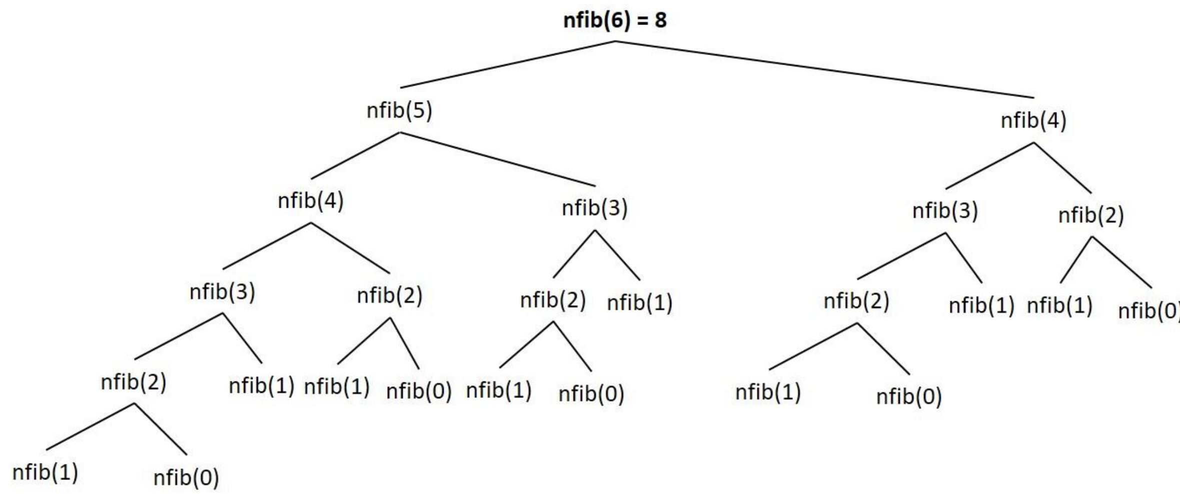

Initialize
Generate 5 sets
Step 1
Process edge (A,D)
Step 2
Process edge (D,E)
Step 3
Process edge (A,C)
Step 4
Process edge (C,B)
A
B
C
D
E
8
10
12
15

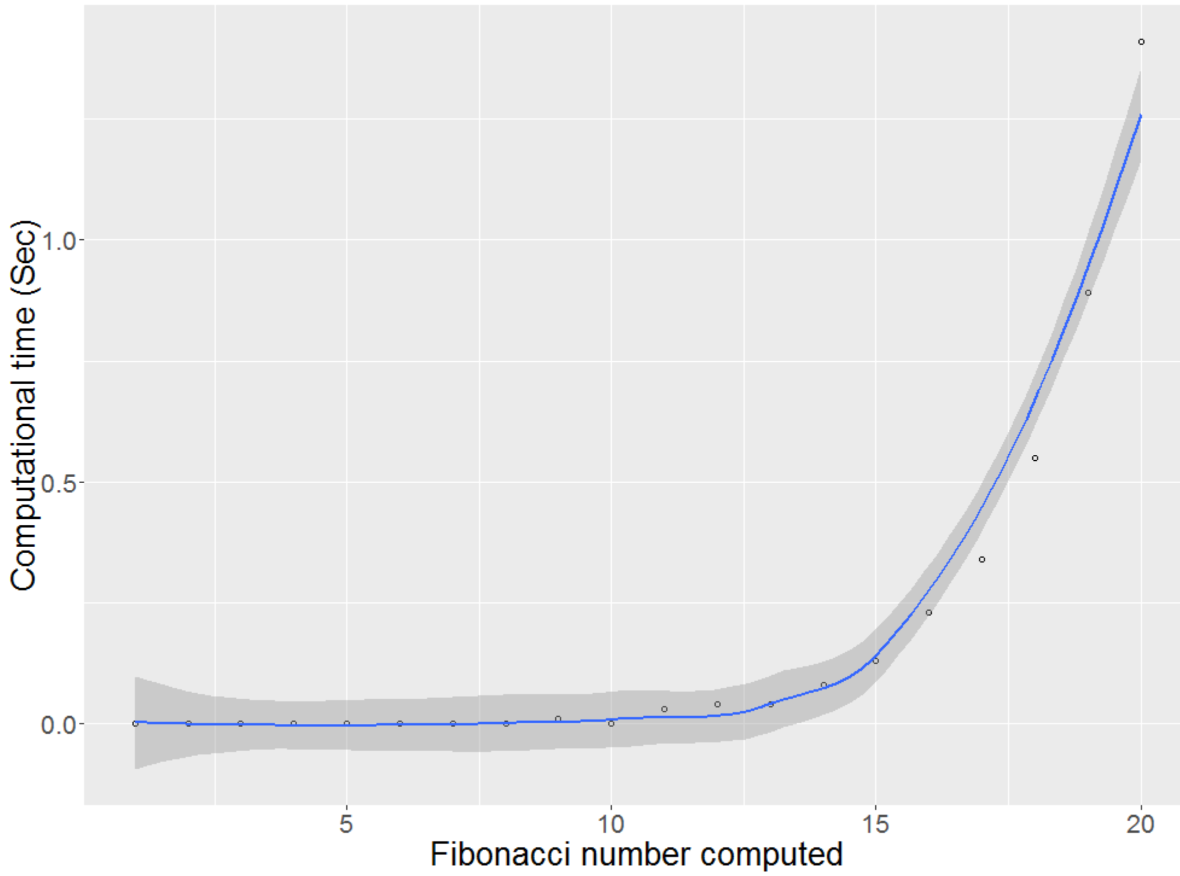

A
B
C
D
E
F
G
7
11
6
6
22
16
9
16
13
7

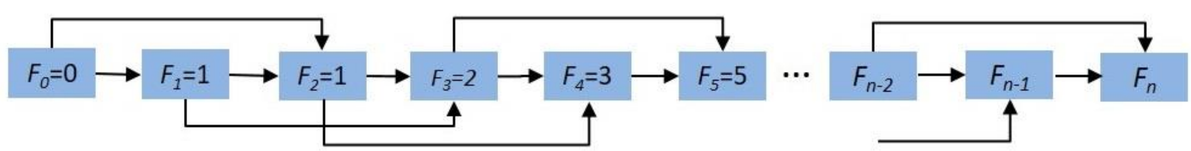
Chapter 9: Programming and Randomized Algorithms





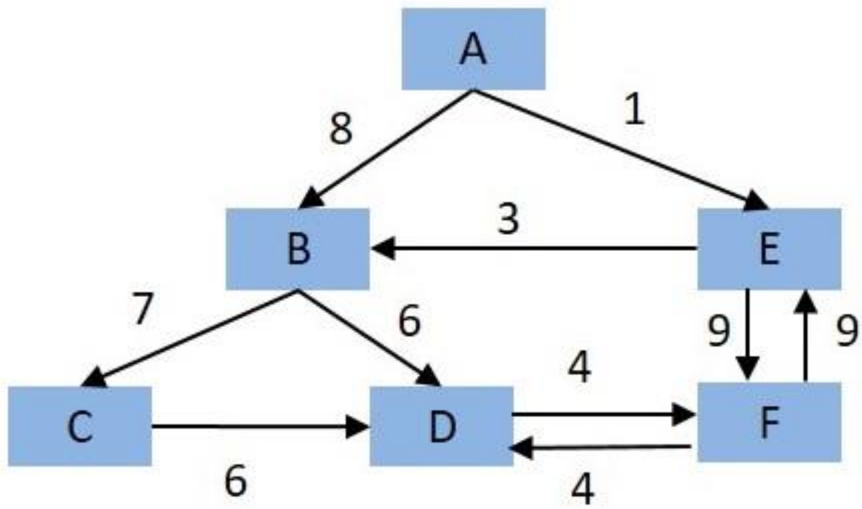

nfib(6) = 8
nfib(5)
nfib(4)
nfib(4)
nfib(3)
nfib(3)
nfib(2)
nfib(3)
nfib(2)
nfib(2)
nfib(1)
nfib(2)
nfib(1)
nfib(1)
nfib(0)
nfib(2)
nfib(1)
nfib(1)
nfib(0)
nfib(1)
nfib(0)
nfib(1)
nfib(0)
nfib(1)
nfib(0)





$$\text{maximize} \sum_{i \in F(*)} c_i$$
$$\text{subject to} \sum_{i \in F(*)} s_i \le W$$

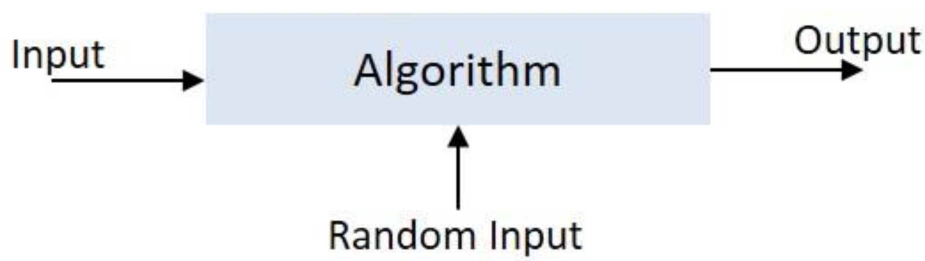
$(u,v) \in E$



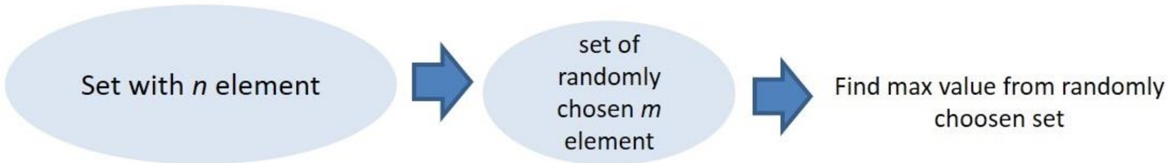
$$d(u,v) = \begin{cases} 0 \ \text{ if } u = v \\ dist(u,v) \ \text{ if } u \neq v \text{ and } (u,v) \in E \\ \infty \text{ else} \end{cases}$$

	A	B	C	D	E	F
A	0	4	11	10	1	10
B	*Inf*	0	7	6	19	10
C	*Inf*	13	0	6	19	10
D	*Inf*	7	14	0	13	4
E	*Inf*	3	10	9	0	9
F	*Inf*	3	10	4	9	0

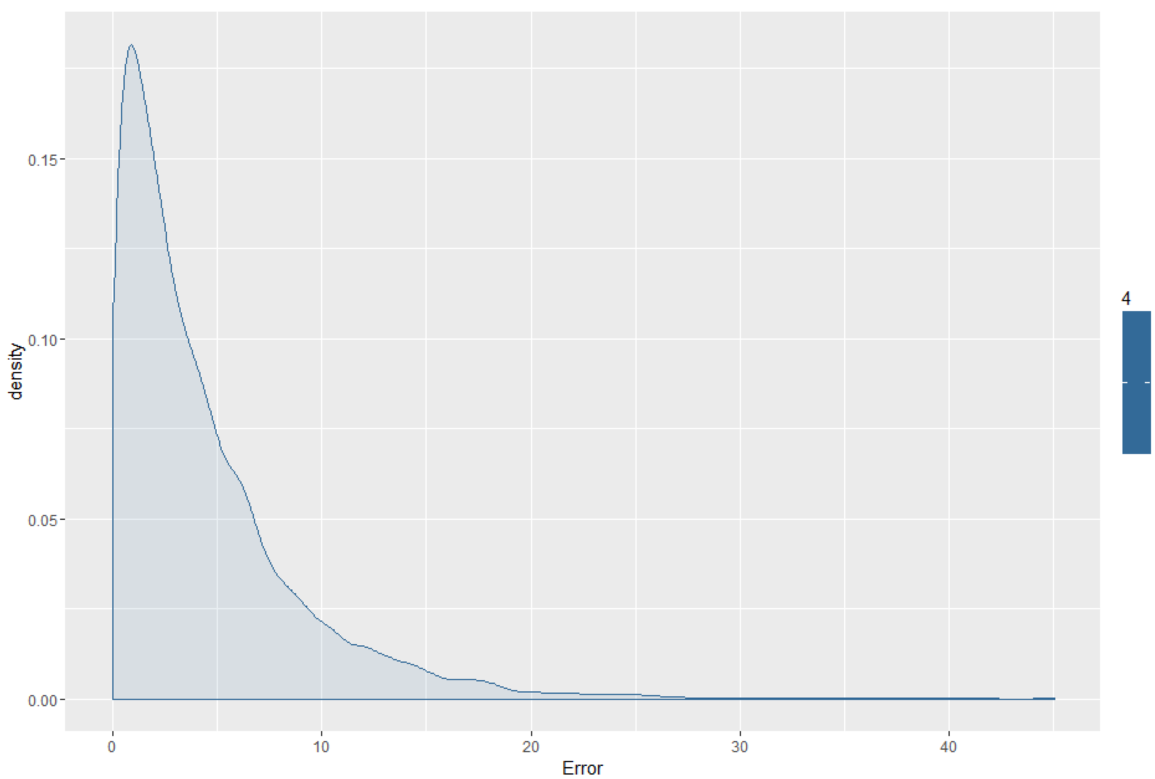
(a) Deterministic algorithm structure



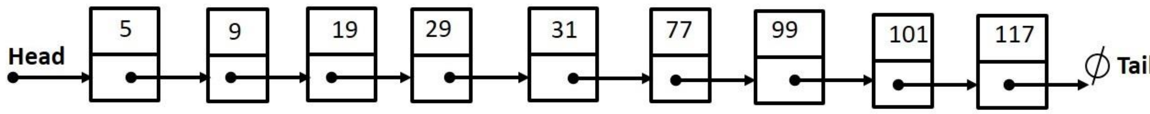
(b) Randomized algorithm structure

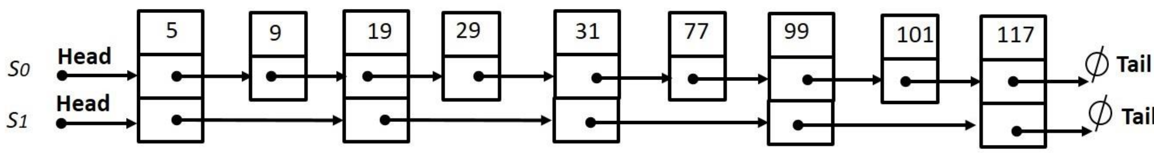


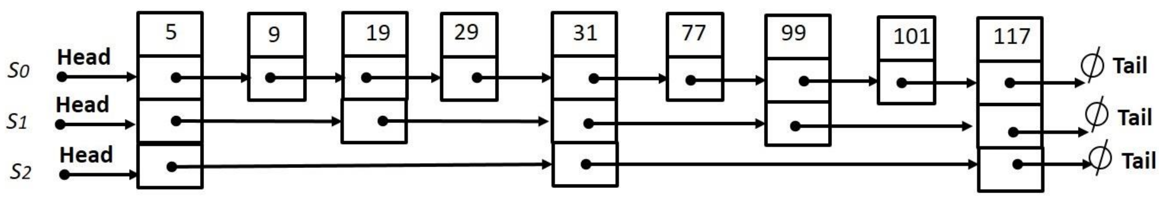
$$1-\left(\frac{19}{20}\right)^{20}$$

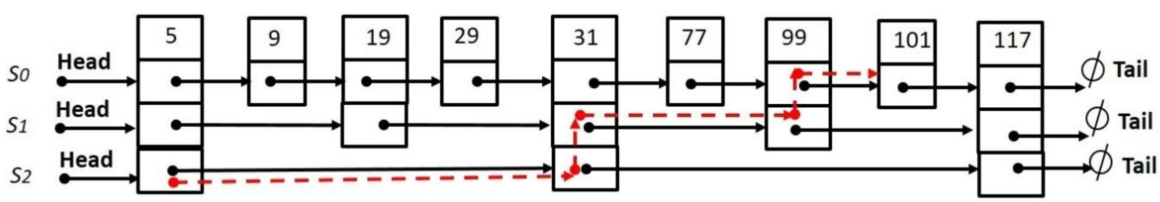


$$1-\frac{1}{2^k}$$

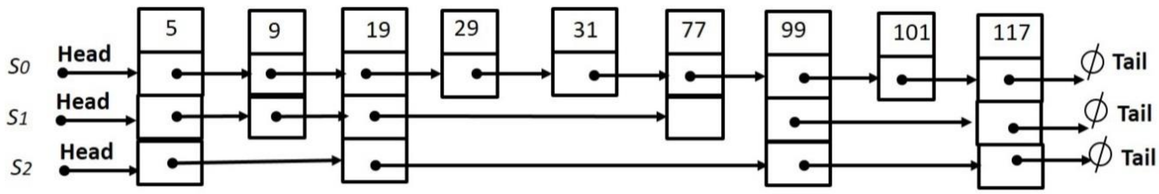

Head
5
9
19
29
31
77
99
101
117
Tail

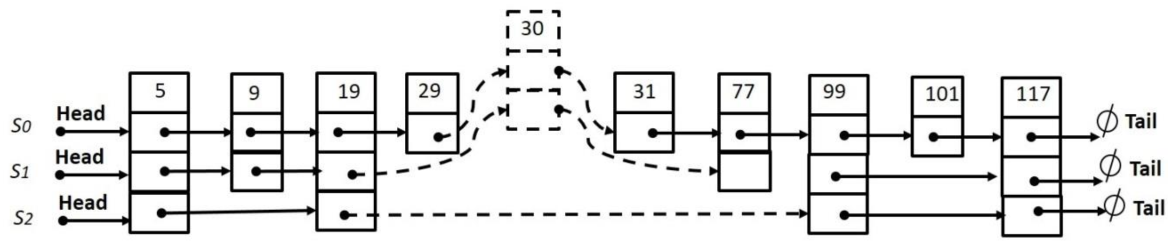

S0
S1
Head
Head
5
9
19
29
31
77
99
101
117
Tail
Tail

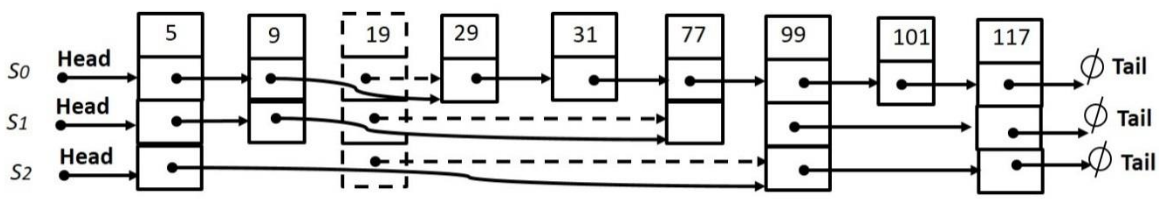

S0
S1
S2
Head
Head
Head
5
9
19
29
31
77
99
101
117
Tail
Tail
Tail


S0
S1
S2
Head
Head
Head
5
9
19
29
31
77
99
101
117
Tail
Tail
Tail

$$\frac{1}{2^k}$$







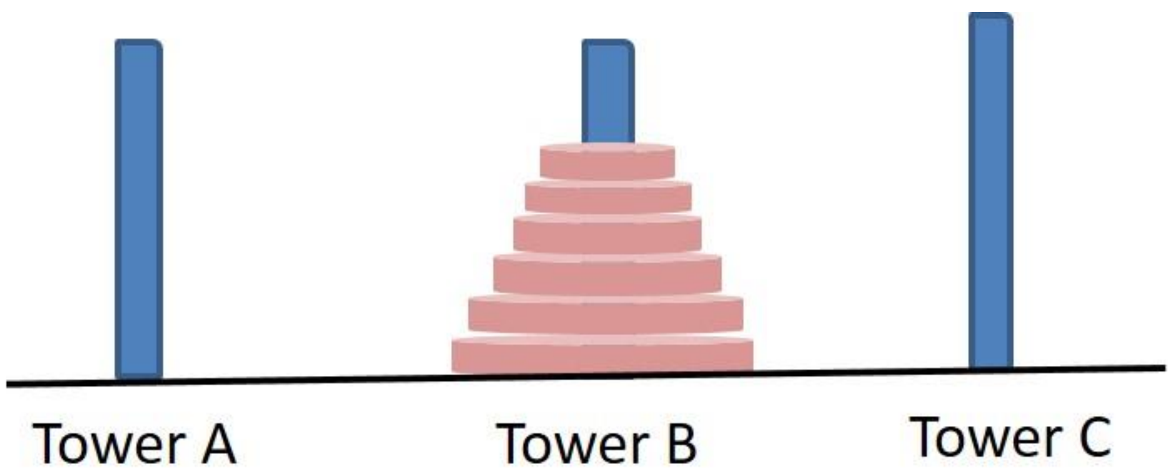
$$\frac{n}{2^k}$$

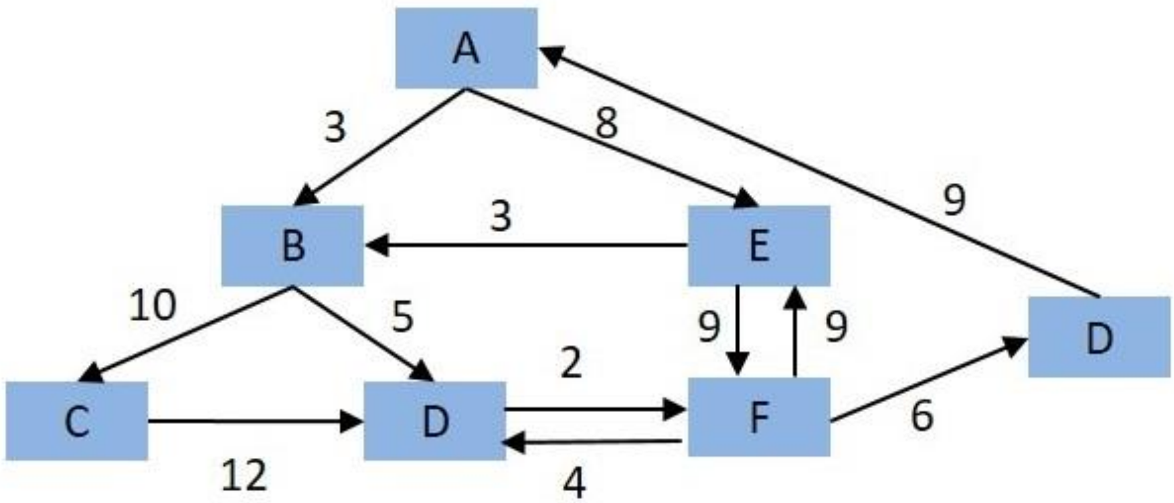
$$P_{3\log n} \leq \frac{n}{2^{3\log n}} = \frac{n}{n^3}$$

$$P_{c\log n} \leq \frac{n}{2^{c\log n}} = \frac{n}{n^c} = \frac{1}{n^{c-1}}$$

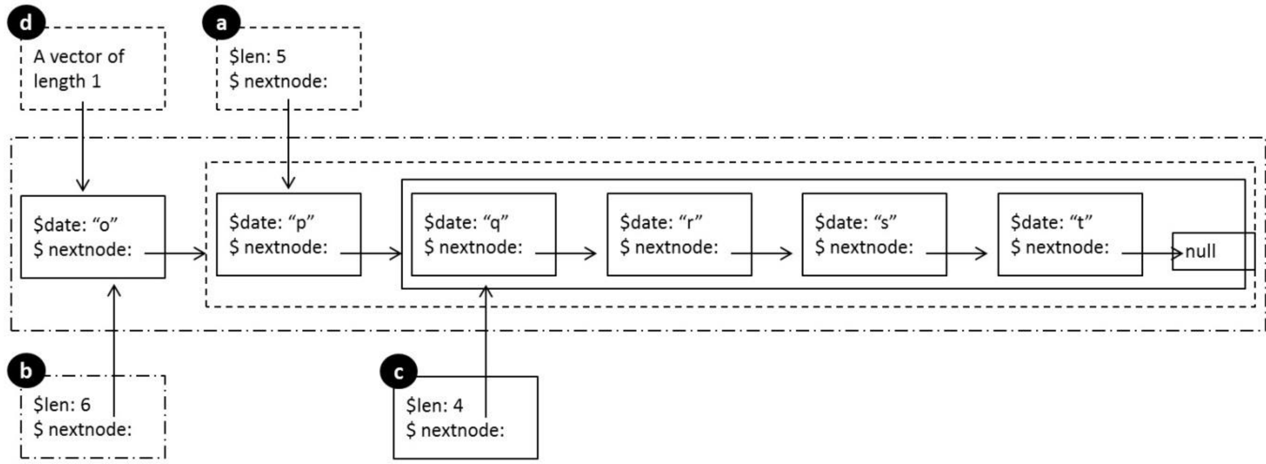
$$\sum_{k=1}^{h} \frac{n}{2^k} = n \sum_{k=1}^{h} \frac{1}{2^k}$$

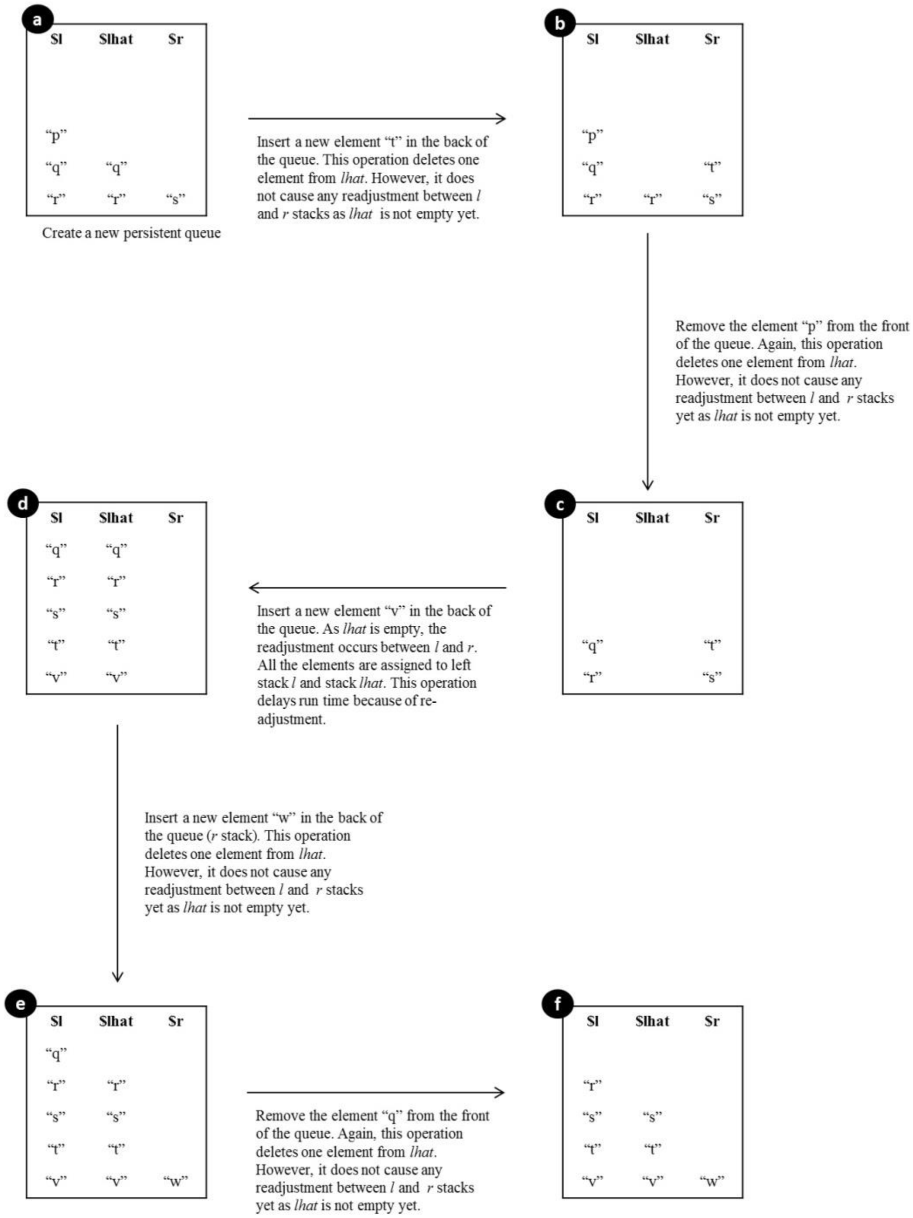
$$\sum\nolimits_{k}^{h} = 1 \frac{1}{2^k} = \frac{\left(\frac{1}{2}\right)^{h+1} - 1}{\frac{1}{2} - 1} = 2\left(1 - \frac{1}{2^{h+1}}\right) < 2 \; for \; all \; n \geq 0$$

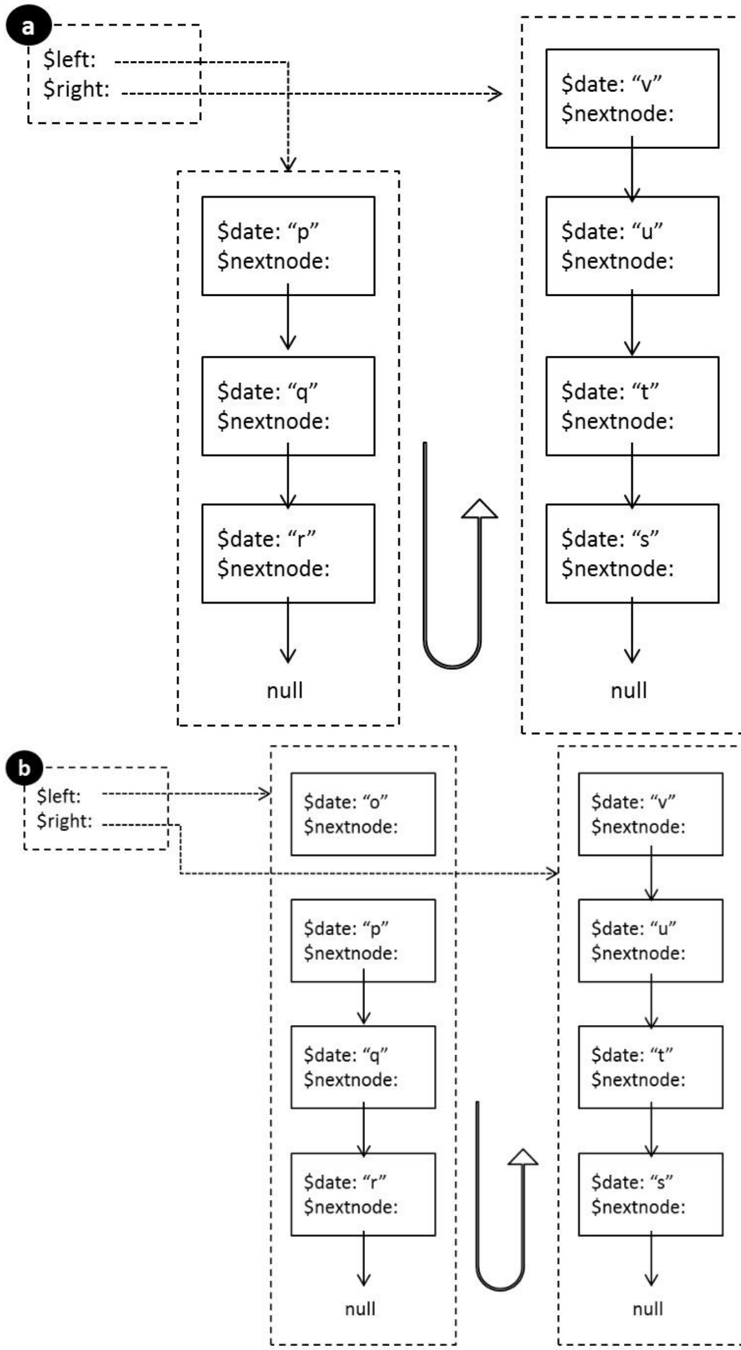


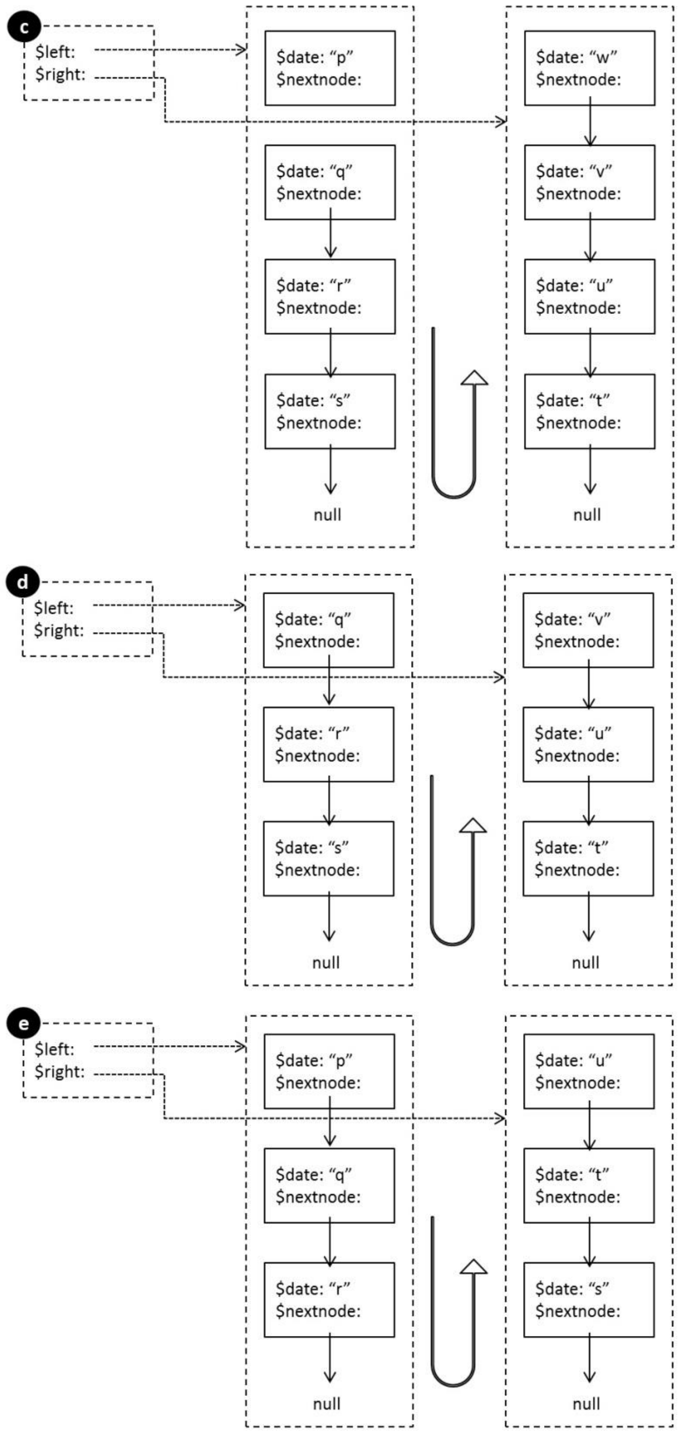

A
3
8
9
B
3
E
10
5
9
9
D
2
C
D
F
6
12
4

Chapter 10: Functional Data Structures




a
Sl
Slhat
Sr
"p"
"q" "q"
"r" "r" "s"
Create a new persistent queue
Insert a new element "t" in the back of the queue. This operation deletes one element from *lhat*. However, it does not cause any readjustment between *l* and *r* stacks as *lhat* is not empty yet.
b
Sl
Slhat
Sr
"p"
"q" "t"
"r" "r" "s"
Remove the element "p" from the front of the queue. Again, this operation deletes one element from *lhat*. However, it does not cause any readjustment between *l* and *r* stacks yet as *lhat* is not empty yet.
c
Sl
Slhat
Sr
"q" "t"
"r" "s"
Insert a new element "v" in the back of the queue. As *lhat* is empty, the readjustment occurs between *l* and *r*. All the elements are assigned to left stack *l* and stack *lhat*. This operation delays run time because of re-adjustment.
d
Sl
Slhat
Sr
"q" "q"
"r" "r"
"s" "s"
"t" "t"
"v" "v"
Insert a new element "w" in the back of the queue (*r* stack). This operation deletes one element from *lhat*. However, it does not cause any readjustment between *l* and *r* stacks yet as *lhat* is not empty yet.
e
Sl
Slhat
Sr
"q"
"r" "r"
"s" "s"
"t" "t"
"v" "v" "w"
Remove the element "q" from the front of the queue. Again, this operation deletes one element from *lhat*. However, it does not cause any readjustment between *l* and *r* stacks yet as *lhat* is not empty yet.
f
Sl
Slhat
Sr
"r"
"s" "s"
"t" "t"
"v" "v" "w"


a
$left:
$right:
$date: "v"
$nextnode:
$date: "u"
$nextnode:
$date: "t"
$nextnode:
$date: "s"
$nextnode:
null
$date: "p"
$nextnode:
$date: "q"
$nextnode:
$date: "r"
$nextnode:
null
b
$left:
$right:
$date: "o"
$nextnode:
$date: "p"
$nextnode:
$date: "q"
$nextnode:
$date: "r"
$nextnode:
null
$date: "v"
$nextnode:
$date: "u"
$nextnode:
$date: "t"
$nextnode:
$date: "s"
$nextnode:
null


c
$left:
$right:
$date: "p" $nextnode:
$date: "q" $nextnode:
$date: "r" $nextnode:
$date: "s" $nextnode:
null
$date: "w" $nextnode:
$date: "v" $nextnode:
$date: "u" $nextnode:
$date: "t" $nextnode:
null
d
$left:
$right:
$date: "q" $nextnode:
$date: "r" $nextnode:
$date: "s" $nextnode:
null
$date: "v" $nextnode:
$date: "u" $nextnode:
$date: "t" $nextnode:
null
e
$left:
$right:
$date: "p" $nextnode:
$date: "q" $nextnode:
$date: "r" $nextnode:
null
$date: "u" $nextnode:
$date: "t" $nextnode:
$date: "s" $nextnode:
null