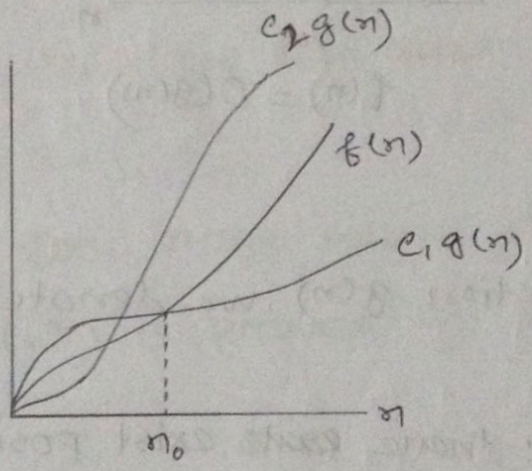
COMPLEXITY ANALYSIS

Θ Notation

For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \}$$

we say that $g(n)$ is an asymptotically tight bound for $f(n)$.



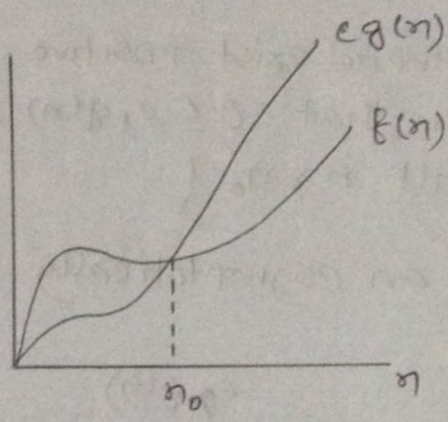
$$f(n) = \Theta(g(n))$$

O Notation

For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$

We say that $g(n)$ is an asymptotically upper bound for $f(n)$.



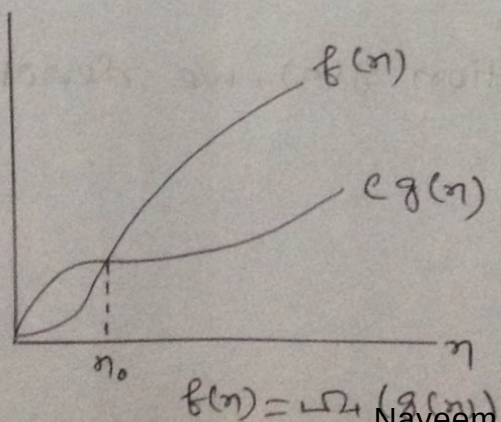
$f(n) = O(g(n))$

Ω Notation

For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$\Omega(g(n)) = \{ f(n)$: there ~~exits~~ exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0 \}$

We say that $g(n)$ is an asymptotically lower bound for $f(n)$



$f(n) = \Omega(g(n))$

Show that $an^2+bn+c \neq \Theta(n^3)$

For the purpose of contradiction, let us assume that

$$an^2+bn+c = \Theta(n^3)$$

This implies that there exist positive numbers c_1, c_2 and n_0 such that

$$0 \leq c_1 n^3 \leq an^2+bn+c \leq c_2 n^3 \text{ for all } n \geq n_0.$$

So, dividing by n^2, we get

$$c_1 n \leq a + \frac{b}{n} + \frac{c}{n^2} \leq c_2 n$$

The relation on the right side i.e.

$a + \frac{b}{n} + \frac{c}{n^2} \leq c_2 n$ holds for very large values of n. But the left sided relation

$$c_1 n \leq a + \frac{b}{n} + \frac{c}{n^2}$$

doesn't hold for large values of n. Hence, $an^2+bn+c \neq \Theta(n^3)$ is proved by contradiction.

Show that $4n^3+5n \neq \Theta(n^2)$

For the purpose of contradiction, let use assume that

$$4n^3+5n = \Theta(n^2)$$

This implies that there exist positive numbers c_1, c_2 and n_0 such that

$$0 \leq c_1 n^2 \leq 4n^3+5n \leq c_2 n^2 \text{ for all } n \geq n_0.$$

So, dividing by n^2, we get

$$c_1 \leq 4n + \frac{5}{n} \leq c_2$$

The relation on left side i.e. $c_1 \leq 4n + \frac{5}{n}$ holds for large values of n.

But the relation on right side i.e. $4n + \frac{5}{n} \leq c_2$ doesn't hold for large values of n since c_2 is a constant.

So, our assumption is not true at all. Hence it is proved by contradiction that

$$4n^3 + 5n \neq \theta(n^2)$$

Nayeem Mahmood / 4AM / C161026

Dynamic Programming

⊞ Write down the sequences of 4 steps to develop a dynamic programming.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

⊞ What do you mean by optimization problems? Which of the following are optimization problems? Give justification in support of your answer.

i) Choosing a path to go from Dhaka to Seoul so that no sea is crossed.
ii) Finding a way to go from Chittagong to Rangpur with minimum cost.
iii) Finding a way to best utilize the class rooms of the department.
iv) Choosing the maximum from a list of numbers.
v) Finding an empty class room from the list of class rooms.

~~vi)~~

Optimization Problem: Those problems are called optimization problems which can have many possible solution having a value and someone wish to find a solution with the optimal (minimum or maximum) value.

Nayeem Mahmood / 4AM / C161026

i) Not an optimization problem since the problem doesn't exhibits optimal substructure property.

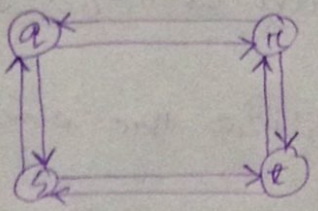
ii) It's an optimization problem since it can be divided into sub-problems.

iii) It's an optimization problem. There exists more than one possible solution.

iv) Not an optimization problem. Exact solution is possible.

v) It's an optimization problem. More than one solution is possible.

III What is optimal substructure? Taking the following graph as an example, show that 'longest simple path' problem doesn't have optimal substructure property.



Optimal sub-structure: If an optimal solution to a problem contains within it optimal solutions to its sub-problems, then it is said that the given problem exhibits optimal substructure property.

The path $q \to r \to t$ is a longest simple path from q to t. But neither the sub-path $q \to r$ is not a longest simple path from q to r, nor is the sub-path $r \to t$ a longest simple path from r to t.

Nayeem Mahmood / 4AM / C161026

Since the supposed optimal solution doesn't contain itself the solutions to its sub-problems, the given 'longest simple path' doesn't have optimal sub-structure property.

⊞ Write down two similarities and two dissimilarities between divide and conquer and dynamic programming paradigms.

Similarities:

1. Both paradigms divide a given problem into some sub-problem and solve the sub-problems.
2. Both paradigms combine the solutions to the sub-problems to obtain the final solution.

Dissimilarities:

1. In dynamic programming, sub-problems may be overlapped and those overlapped sub-problems are solved once and then stored in a table for future look up. But in divide and conquer paradigm, sub-problems may or may not be ~~occurred~~ overlapped. If does, it solves them everytime i.e. it doesn't use memoization.
2. In divide and conquer, the sub-problems are independent of each other while in case of dynamic programming, the sub-problems are not independent ~~in~~ of each other i.e. solution to one sub-problem may be required to solve another sub-problem.

What do you mean by overlapping sub-problems?

When a recursive algorithm revisits the same problem repeatedly, it is said that the optimization problem has overlapping sub-problems.

Why Divide and Conquer is applied instead of DP when a problem doesn't have overlapping sub-problem property?

In dynamic programming, a sub-problem is solved as a part of one bigger sub-problem which might be required to be solved again as a part of another subproblem. So, the very first time, a sub-problem is solved and it's result are tabulated/stored and the next time it is encountered, instead of solving it again, the result is looked up in the table using constant time per lookup.

Since, memoization isn't needed when a problem doesn't have overlapping sub-problem property, divide and conquer is used to solve it instead of DP.

Nayeem Mahmood / 4AM / C161026

Consider the following function :

$$F(n) = F(\lfloor n/2 \rfloor) \times F(\lfloor n/4 \rfloor) \quad ; n > 0$$

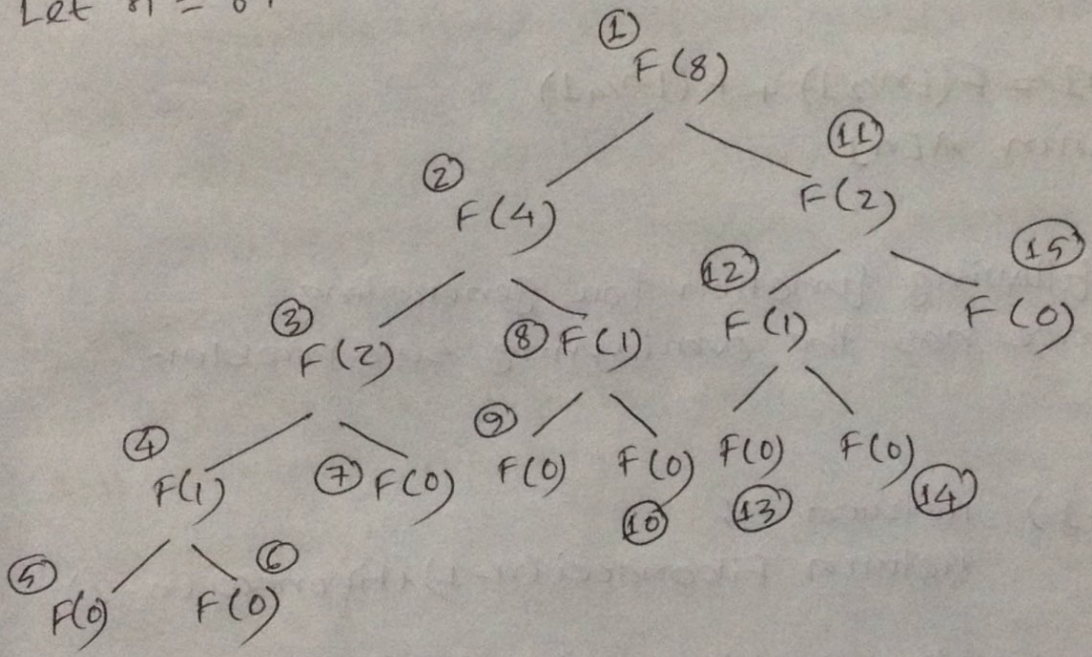
$$F(n) = 1 \quad ; n = 0$$

i) Why should you use DP to evaluate this function?
ii) Write a dynamic programming solution/algorithm to evaluate this function.

---

i) To better understand the usefulness of DP to solve the function, let us assume a value for n and examine the recursive calling tree.

Let $n = 8$.



We can see that there is total 15 function calls for $n = 8$. If we examine, it is noticeable that number 4, 8 and 12 are the same calling. That applies for number 3 and 11. These callings are redundant. What if we could save the result for F(2) that was got from number 3 ? In that case, we don't have to calculate F(2) again while it is called again in number 11 calling. Hence, we should use DP to evaluate

this function.

ii)

Let A[1..n] be an array which can be accessed globally and is set 0 to all the elements initially.

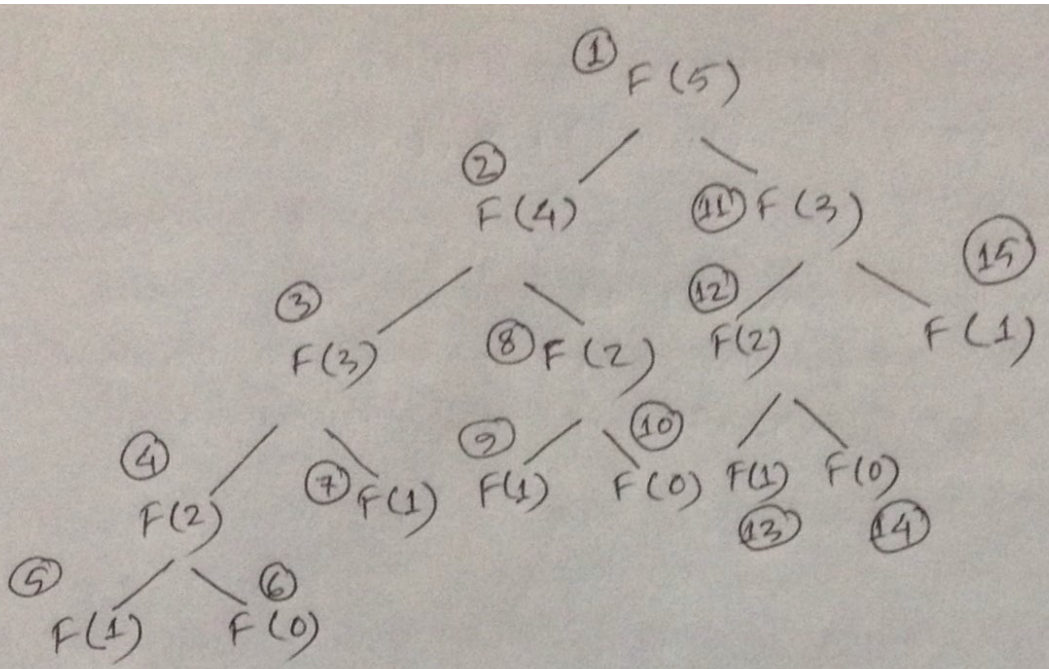
```
F(n):
1.  if n == 0
2.        return 1
3.  else if A[n] != 0
4.        return A[n]
5.  else
6.        A[n] = F(⌊n/2⌋) + F(⌊n/4⌋)
7.        return A[n]
```

Show that the following function for generating Fiboracci sequence has the overlapping sub-problem property.

```
Fiboracci(n)
        if (n <= 1)   return n;
        else          return Fiboracci(n-1) + Fiboracci(n-2);
```

How can you solve the problem using DP?

For n = 5, the recursive callings tree be like below;



We can notice that F(2) is called thrice (function call number: 4, 8, 12). Hence according to definition and the recursive tree above, the given problem has overlapping sub-problem property.

DP Soln:

Let A[1..n] be a global array and is set 0 to all the elements initially.

```
Fibonacci(n):
1. if n <= 1
2.         return n
3. else if A[n] != 0
4.         return A[n]
5. else
6.         A[n] = Fibonacci(n-1) + Fibonacci(n-2)
7.         return A[n]
```

Nayeem Mahmood / 4AM / C161026

Matrix Chain Multiplication

The required function to calculate the minimum cost

$$m[i,j] = \begin{cases} 0 & ; i == j \\ \min\limits_{i \le k < j} \{ m[i,k] + m[k+1, j] + P_{i-1} P_k P_j \} & ; i < j \end{cases}$$

Given a sequence $P = \{3, 2, 4, 5, 6\}$ which represents the chain of matrices such that i-th matrix A_i is of dimension $P_{i-1} \times P_i$. Fill up the following m and s tables for computing the cost of matrix chain multiplication using dynamic programming for the matrix chain A.

matrix	A_1	A_2	A_3	A_4
dimension	3×2	2×4	4×5	5×6
	$P_0\ P_1$	$P_1\ P_2$	$P_2\ P_3$	$P_3\ P_4$

m

j \ i	1	2	3	4
4	136	100	120	0
3	70	40	0	
2	24	0		
1	0			

s

j \ i	1	2	3
4	1	3	3
3	1	2	
2	1		

$m[1,2]$ →

When $k = 1$,

$$m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$$

$$= 0 + 0 + (3 \times 2 \times 4)$$

$$= 24$$

and $s[1,2] = 1$

Nayeem Mahmood / 4AM / C161026

$m[2,3]$ →

When $k=2$,

$$m[2,3] = m[2,2] + m[3,3] + p_1 p_2 p_3$$
$$= 0 + 0 + (2 \times 4 \times 5)$$
$$= 40$$

and $s[2,3] = 2 \quad [\because k=2]$

$m[3,4]$ →

When $k=3$,

$$m[3,4] = m[3,3] + m[4,4] + p_2 p_3 p_4$$
$$= 0 + 0 + (4 \times 5 \times 6)$$
$$= 120$$

and $s[3,4] = 3$

$m[1,3]$ →

When $k=1$,

$$m[1,3] = m[1,1] + m[2,3] + p_0 p_1 p_3$$
$$= 0 + 40 + (3 \times 2 \times 5)$$
$$= 70$$

When $k=2$,

$$m[1,3] = m[1,2] + m[3,3] + p_0 p_2 p_3$$
$$= 24 + 0 + (3 \times 4 \times 5)$$
$$= 84$$

$\therefore m[1,3] = 70$ when $k=1$

and $s[1,3] = 1$

$m[2,4]$ →

when $k=2$,

$$m[2,4] = m[2,2] + m[3,4] + p_1 p_2 p_4$$
$$= 0 + 120 + (2 \times 4 \times 6)$$
$$= 168$$

when $k=3$,

$$m[2,4] = m[2,3] + m[4,4] + p_1 p_3 p_4$$
$$= 40 + 0 + (2 \times 5 \times 6)$$
$$= 100$$

$\therefore m[2,4] = 100$

and $s[2,4] = 3$

$m[1,4]$ →

when $k=1$,

$$m[1,4] = m[1,1] + m[2,4] + p_0 p_1 p_4$$
$$= 0 + 100 + (3 \times 2 \times 6)$$
$$= 136$$

when $k=2$,

$$m[1,4] = m[1,2] + m[3,4] + p_0 p_2 p_4$$
$$= 24 + 120 + (3 \times 4 \times 6)$$
$$= 216$$

when $k=3$,

$$m[1,4] = m[1,3] + m[4,4] + p_0 p_3 p_4$$
$$= 70 + 0 + (3 \times 5 \times 6)$$

$= 160$

$\therefore m[1,4] = 136$

and $s[1,4] = 1$

Hence, the minimal cost is $m[1,4] = 136$.

Optimal Parenthesization for the given matrices:

$$(A_1)((A_2 \times A_3)(A_4))$$

[+] Recursively express the cost of multiplying a chain of matrices.

The recursive expression is

$$m[i,j] = \begin{cases} 0 & ; i == j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j & ; i < j \end{cases}$$

[+] Given a sequence $p = \{2,2,3,3,5\}$ which represents the chain of matrices such that i-th matrix A_i is of dimension $p_{i-1} \times p_i$

i) Fill up m and s tables.

ii) Why the table s is needed?

i)

matrix	A_1	A_2	A_3	A_4
dimension	2×2	2×3	3×3	3×5
	P_0 P_1	P_1 P_2	P_2 P_3	P_3 P_4

m

i \ j	1	2	3	4
4	60	48	45	0
3	30	18	0	
2	12	0		
1	0			

s

i \ j	1	2	3
4	3	3	3
3	1	2	
2	1		

m[1,3] →

when k = 1,

$$m[1,3] = m[1,1] + m[2,3] + P_0P_1P_3$$
$$= 0 + 18 + (2 \times 2 \times 3)$$
$$= 30$$

when k = 2,

$$m[1,3] = m[1,2] + m[3,3] + P_0P_2P_3$$
$$= 12 + 0 + (2 \times 3 \times 3)$$
$$= 30$$

$\therefore m[1,3] = 30$ and $s[1,3] = 1$

m[2,4] →

when k = 2,

$$m[2,4] = m[2,2] + m[3,4] + P_1P_2P_4$$
$$= 0 + 45 + (2 \times 3 \times 5)$$
$$= 75$$

Nayeem Mahmood / 4AM / C161026

When $k = 3$,

$$m[2,4] = m[2,3] + m[4,4] + P_1 P_3 P_4$$
$$= 18 + 0 + (2 \times 3 \times 5)$$
$$= 48$$

$\therefore m[2,4] = 48$ and $s[2,4] = 3$

$m[1,4]$ →

when $k = 1$,

$$m[1,4] = m[1,1] + m[2,4] + P_0 P_1 P_4$$
$$= 0 + 48 + (2 \times 2 \times 5)$$
$$= 68$$

when $k = 2$,

$$m[1,4] = m[1,2] + m[3,4] + P_0 P_2 P_4$$
$$= 12 + 45 + (2 \times 3 \times 5)$$
$$= 87$$

when $k = 3$,

$$m[1,4] = m[1,3] + m[4,4] + P_0 P_3 P_4$$
$$= 30 + 0 + (2 \times 3 \times 5)$$
$$= 60$$

$\therefore m[1,4] = 60$ and $s[1,4] = 3$

Hence, the optimal cost is $m[1,4] = 60$.

Optimal Parenthesization

$$((A_1)\ (A_2 \times A_3))(A_4)$$

ii)

The s table records the value for k when m[i, j] is minimum. From that table, we can easily determine where to put parentheses.

[#] Given a sequence $P = \{30, 35, 15, 5, 10, 20, 25\}$ which represents the chain of matrices such that i-th matrix A_i is of dimension $P_{i-1} \times P_i$.

i) Calculate m[3, 6] and s[3, 6].

ii) Find an optimal parenthesization of matrix chain product for above matrices.

---

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25
	$P_0\ P_1$	$P_1\ P_2$	$P_2\ P_3$	$P_3\ P_4$	$P_4\ P_5$	$P_5\ P_6$

m

j \ i	1	2	3	4	5	6
6	15,125	10,500	5,375	3,500	5,000	0
5	11,875	7,125	2,500	1,000	0	
4	9,375	4,375	750	0		
3	7,875	2,625	0			
2	15,750	0				
1	0					

Nayeem Mahmood / 4AM / C161026

	1	2	3	4	5
6	3	3	3	5	5
5	3	3	3	4	
4	3	3	3		
3	1	2			
2	1				

i)

$m[3,6]$ ⟶

when $k=3$,

$$m[3,6] = m[3,3] + m[4,6] + P_2P_3P_6$$
$$= 0 + 3500 + (15 \times 5 \times 25)$$
$$= 5375$$

when $k=4$,

$$m[3,6] = m[3,4] + m[5,6] + P_2P_4P_6$$
$$= 750 + 5000 + (15 \times 10 \times 25)$$
$$= 9500$$

when $k=5$,

$$m[3,6] = m[3,5] + m[6,6] + P_2P_5P_6$$
$$= 2500 + 0 + (15 \times 20 \times 25)$$
$$= 10000$$

$\therefore m[3,6] = 5375$ and

$s[3,6] = 3$

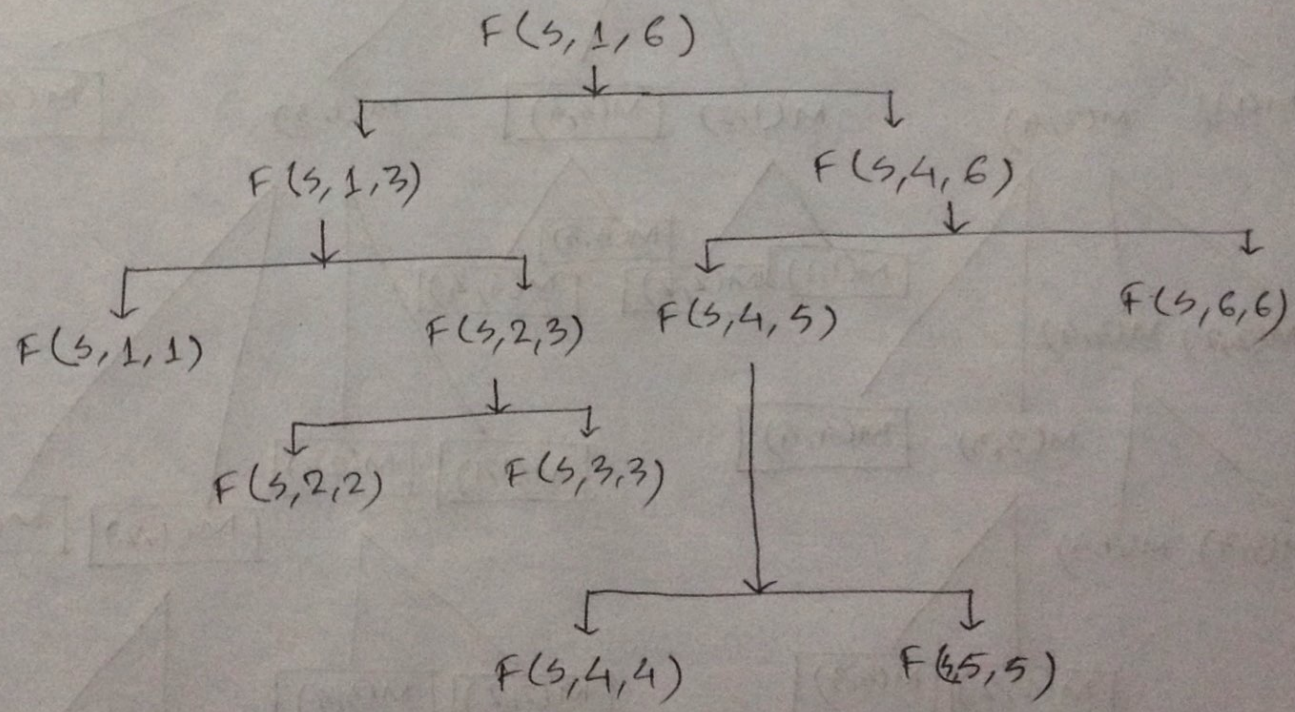
ii)

$$\Big(\big(A_1 \quad (A_2 \times A_3)\big)\big((A_4 \times A_5) \quad A_6\big)\Big)$$

⊛ The recursive procedure to find parenthesization

```
F(s, i, j):
1.    if i == j
2.        print "Ai"
3.    else
4.        print "("
5.        F(s, i, s[i, j])
6.        F(s, s[i, j] + 1, j)
7.        print ")"
```

The recursive calling tree be

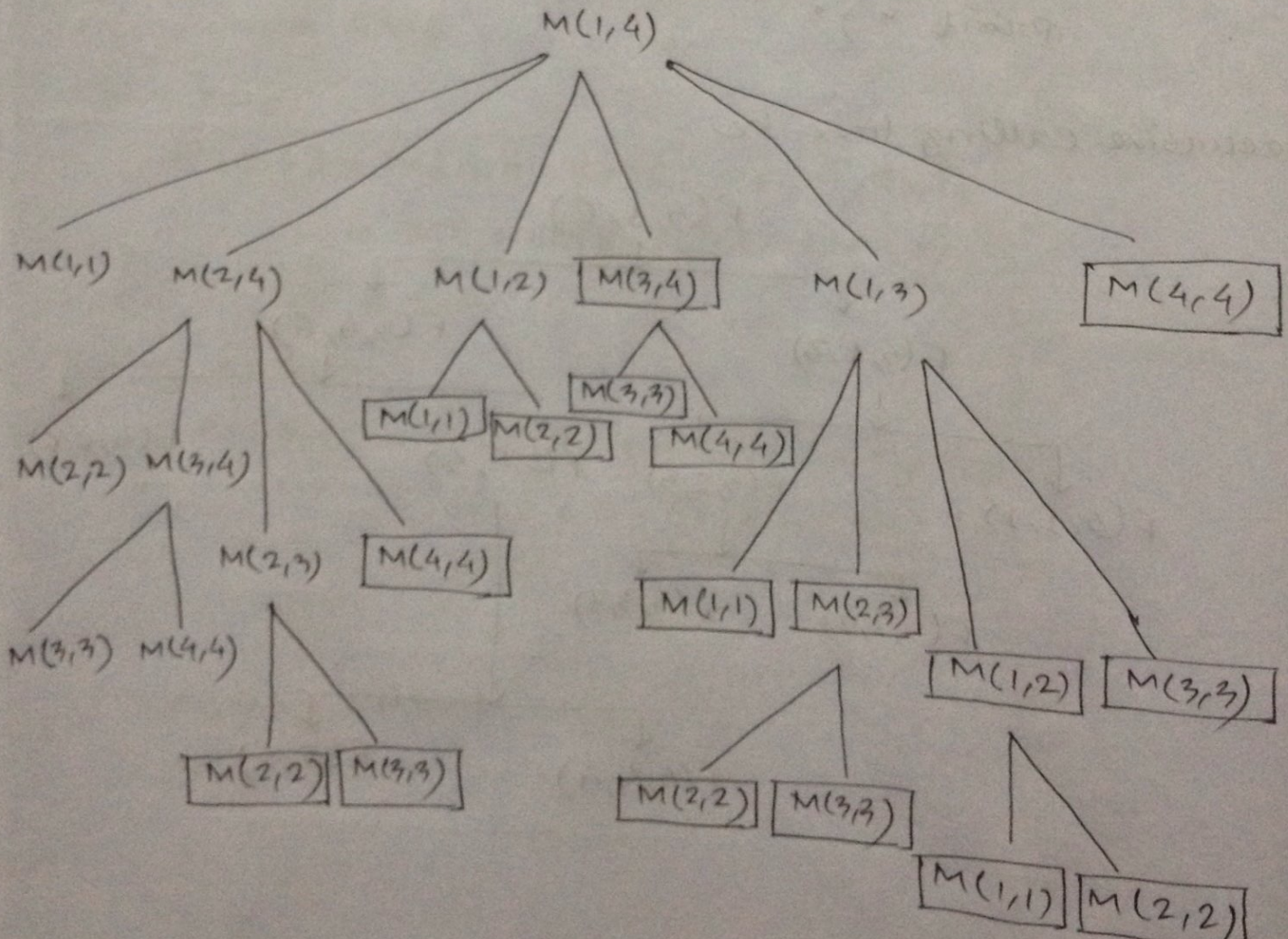


Suppose, you need to find the optimal parenthesization of a matrix chain product. Find a recursive solution and write the appropriate equation(s).

[see above]

Show that the matrix chain multiplication problem has overlapping sub-problem property.

Let us consider M(1, n) be a recursive procedure which calculates the minimum number of scalar multiplications needed for a matrix chain $\langle A_1, A_2, \ldots A_n \rangle$. The the recursive calling tree for M(1,4) be



Nayeem Mahmood / 4AM / C161026

The redundant callings are boxed. We can see that M(1,2) was called for finding solutions for both M(1,4) and M(1,3).

Hence, we can say that this problem has overlapping sub-problem property.

Nayeem Mahmood / 4AM / C161026

Longest Common Subsequence

1. Write down the recursive formula for the optimal sub-structure of the LCS problem.

$$c[i,j] = \begin{cases} 0 & ; \text{if } i=0 \text{ or } j=0 \\ c[i-1, j-1] + 1 & ; \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & ; \text{if } j>0 \text{ and } x_i \neq y_j \end{cases}$$

2. Find the LCS of two strings X = "ADCFGEFD" and Y = "ACBFHED" by generating LCS table. Using that table, tell the LCS length of ("ADCFGE", "ACBF") and ("ADCFGE", "ACBFHE")

Nayeem Mahmood / 4AM / C161026

Length Table

		0	1	2	3	4	5	6	7
	Y_i		A	C	B	F	H	E	D
0	X_i	0	0	0	0	0	0	0	0
1	A	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1
2	D	0	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1	↖ 2
3	C	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	↑ 2
4	F	0	↑ 1	↑ 2	↑ 2	↖ 3	← 3	← 3	← 3
5	G	0	↑ 1	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
6	E	0	↑ 1	↑ 2	↑ 2	↑ 3	↑ 3	↖ 4	← 4
7	F	0	↑ 1	↑ 2	↑ 2	↑ 3	↑ 3	↑ 4	↑ 4
8	D	0	↑ 1	↑ 2	↑ 2	↑ 3	↑ 3	↑ 4	↖ 5

The LCS of the two given strings is "ACFED" of length 5.

LCS Length of "ADCFGE" and "ACBF"

The cell C[6,4] gives us the length which is 3.
[The LCS is "ACF"]

Lcs length of "ADCFHE" and "ACBFHE"

The cell c[6,6] from above table gives us the length which is 4.

[The Lcs is "ACFE"]

Find Lcs of X = {a, b, a, d, c, b} and Y = {b, c, a, d, c} using DP.

Lcs Table

		0	1	2	3	4	5
	Y_j		(b)	c	(a)	(d)	(c)
0	X_i	0	0	0	0	0	0
1	a	(0)	↑ 0	↑ 0	↖ 1	← 1	← 1
2	(b)	0	(↖ 1)	(← 1)	↑ 1	↑ 1	↑ 1
3	(a)	0	↑ 1	↑ 1	(↖ 2)	← 2	← 2
4	(d)	0	↑ 1	↑ 1	↑ 2	(↖ 3)	← 3
5	(c)	0	↑ 1	↖ 2	↑ 2	↑ 3	(↖ 4)
6	b	0	↖ 1	↑ 2	↑ 2	↑ 3	(↑ 4)

Nayeem Mahmood / 4AM / C161026

The LCS of the given strings is "{b, a, d, c}" of length 4.

Ⅲ X = "ACCFE", Y = "ADCFFE".

i) Generate LCS table.

ii) Using the table, mention what is the LCS of X & Y?

iii) Using the table, can you find the LCS of "ACC" & "ADCFFE"?

LCS Table

		0	1	2	3	4	5	6
	Y_i	(A)	D	(C)	F	(F)	(E)	
0	X_i	(0)	0	0	0	0	0	0
1	(A)	0	(↖ 1)	(← 1)	← 1	← 1	← 1	← 1
2	(C)	0	↑ 1	↑ 1	(↖ 2)	(← 2)	← 2	← 2
3	0	0	↑ 1	↑ 1	↑ 2	(↑ 2)	↑ 2	↑ 2
4	(F)	0	↑ 1	↑ 1	↑ 2	↖ 3	(↖ 3)	← 3
5	(E)	0	↑ 1	↑ 1	↑ 2	↑ 3	↑ 3	(↖ 4)

ii) LCS of X and Y is "ACFE" of length 4.

Nayeem Mahmood / 4AM / C161026

iii) The LCS of "ACO" and "ADCFFE" is "AC" is of length 2 which can be found from the cell c[3,6].

Do you think that the LCS problem has overlapping sub-problem property? Justify your answer using suitable example.

Let us consider two strings be $X = \{x_1, x_2, \cdots, x_m\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$. To find LCS of X and Y, we should examine either one or two sub-problems.

If $x_m = y_n$, we must find an LCS of X_{m-1} and Y_{n-1}. Appending $x_m = y_n$ to this LCS yields LCS of X and Y.

If $x_m \neq y_n$, then we must solve two subproblems:

1. finding an LCS of X_{m-1} and Y
2. finding an LCS of X and Y_{n-1}

Whichever of these two LCSs is longer is an LCS of X and Y. Because these cases exhaust all possibilities.

We can readily see the overlapping sub-problem property in the LCS problem. To find an LCS of X and Y, we may need to find the LCSs of X and Y_{n-1} and of X_{m-1} and Y. But each of these sub-problems has the subproblems of finding an LCS of X_{m-1} and Y_{n-1}. Many other subproblems share subproblems.

QUICK SORT

Algorithm

The key to this algorithm is the Partition procedure, which re-arranges the sub-array $A[p..r]$ in place.

```
PARTITION (A, p, r)
1.  x = A[r]
2.  i = p - 1
3.  for j = p to r - 1
4.      if A[j] ≤ x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.  exchange A[i+1] with A[r]
8.  return i + 1
```

```
QUICKSORT (A, p, r)
1.  if p < r
2.      q = PARTITION (A, p, r)
3.      QUICKSORT (A, p, q-1)
4.      QUICKSORT (A, q+1, r)
```

Nayeem Mahmood / 4AM / C161026

Do you think the Quick sort algorithm follows divide and conquer process? Defend your answer.

Yes. Quick sort algorithm follows Divide-and-Conquer process.

Defense: Here is the three steps.

Divide: It partitions the array $A[p..r]$ into two (possibly empty) sub-arrays $A[p..q-1]$ and $A[q+1..r]$ such that each element of $A[p..q-1]$ is less than or equal to $A[q]$, which in turn, less than or equal to each element of $A[q+1..r]$. It computes the index q as part of this partitioning procedure.

Conquer: It sorts two sub-arrays $A[p..q-1]$ and $A[q+1..r]$ by recursive calls to Quick sort.

Combine: Because the sub-arrays are already sorted, no work is needed to combine them; the entire array $A[p..r]$ is now sorted.

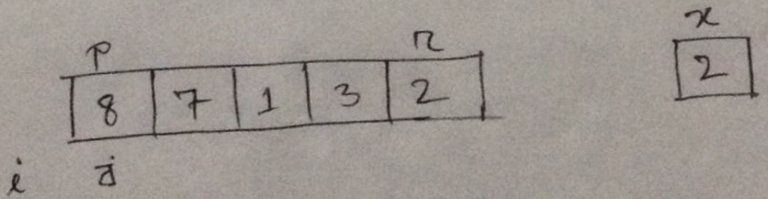
Show the operation of the PARTITION of the Quick sort algorithm on the array

(i) $A = \langle 8, 7, 1, 3, 2 \rangle$

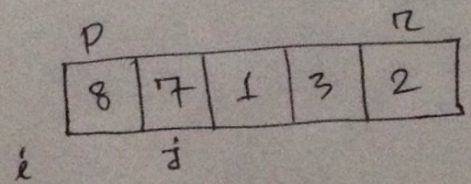
(ii) $A = \langle 8, 1, 7, 3, 2, 4, 6, 5 \rangle$

(i)

(a)

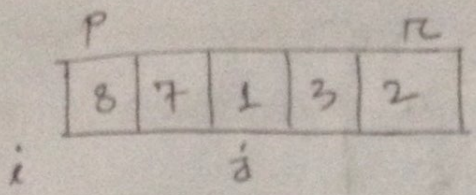


(b)



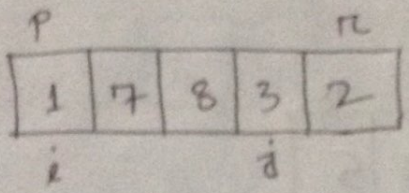
Nayeem Mahmood / 4AM / C161026

(c)

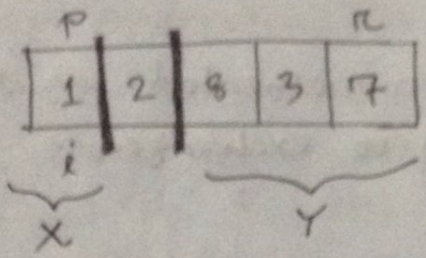


[∵ A[j] ≤ x, then A[i+1] and A[j] will be exchanged]

(d)



(e)

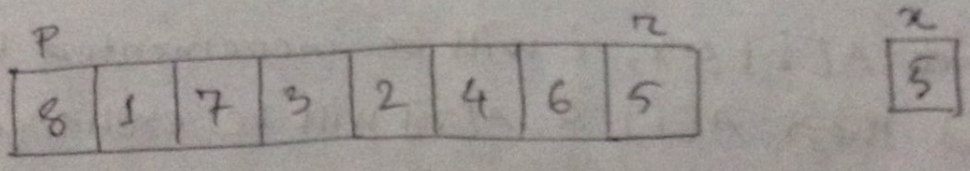


The partition operation on the given array is done. The pivot element A[r] = 2 is in its proper place as we can see that the elements on its left side are less than it and it is less than the elements on its right side.

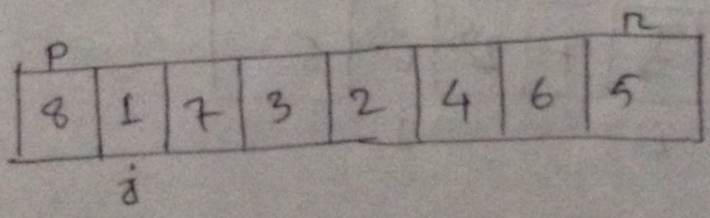
The call to PARTITION Procedure will return 2 as the index of the pivot element is 2 here.

(ii)

(a)

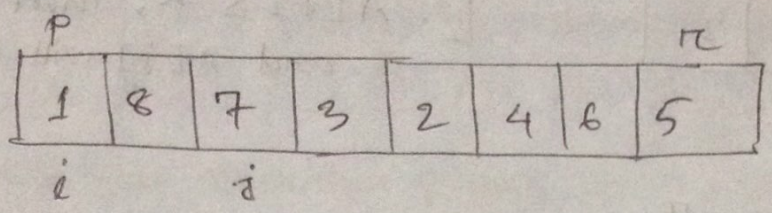


(b)

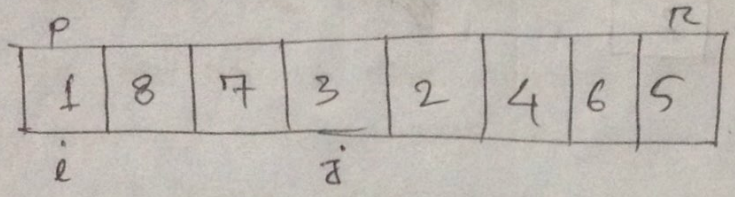


since A[j] ≤ x, A[i+1] and A[j] will be exchanged and i will be incremented by 1

(c)

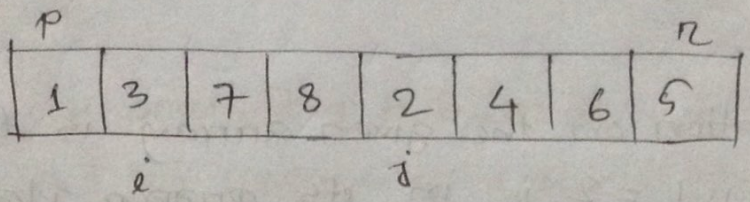


(d)



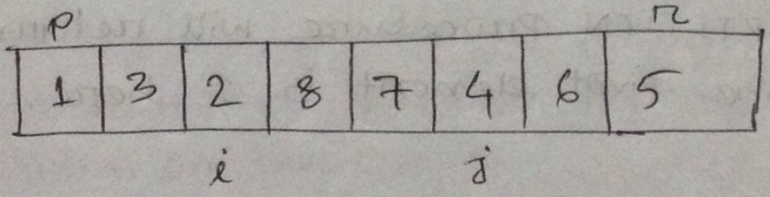
Since $A[j] \leq x$, i will be incremented by 1 and then A[i] and A[j] will be exchanged.

(e)



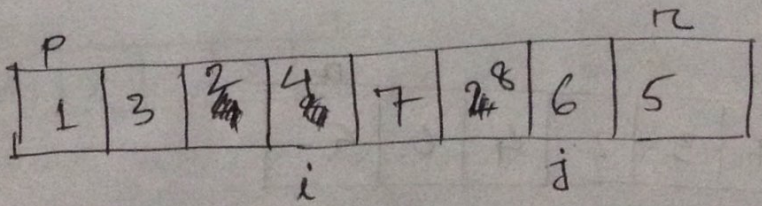
Since $A[j] \leq x$, i will be incremented by 1 and then A[i] and A[j] will be exchanged.

(f)



Since $A[j] \leq x$, i will be incremented by 1 and then A[i] & A[j] will be exchanged.

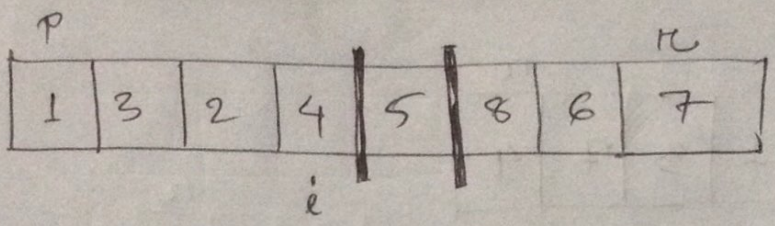
(g)



Now j will be incremented and the loop

will be terminated since $i == r$. Then $A[i+1]$ and $A[r]$ will be exchanged.

(h)



The Partition operation on the given array is done. Pivot element i.e. $x = 5$ is placed in its proper position since the elements on its left are less than or equal to it and it is less than or equal to the elements on its right side.

The call will return $(i+1)$ i.e 5 as the index of pivot is 5 now.

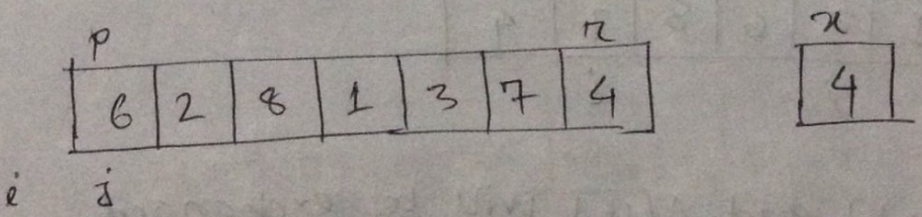
Show the steps of the first PARTITION operation of the quick sort algorithm on the array

(i) $A = \langle 6, 2, 8, 1, 3, 7, 4 \rangle$

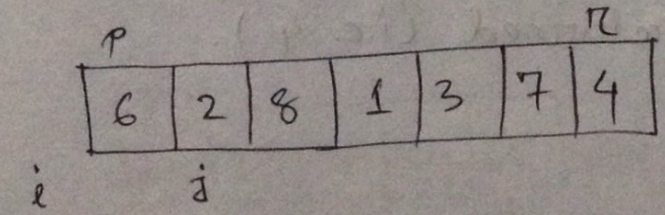
(ii) $A = \langle 14, 19, 54, 20, 65, 37, 17, 85, 95, 32, 45 \rangle$

(i)

(a)

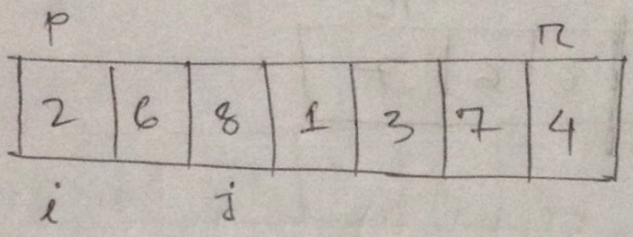


(b)

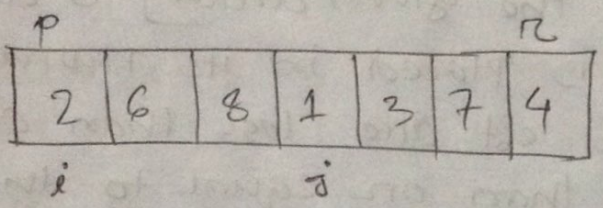


Since, $A[j] \leq x$, i will be incremented and then $A[i]$ and $A[j]$ will be swapped.

(c)

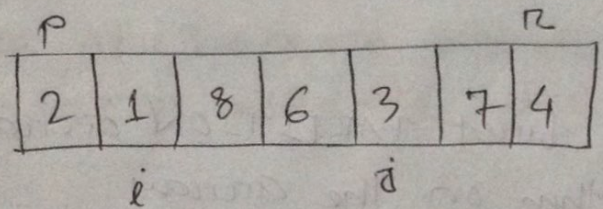


(d)



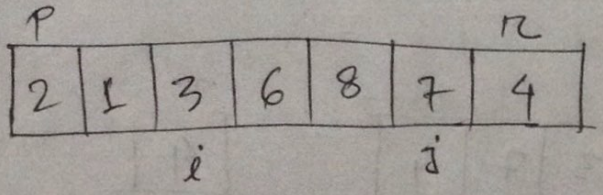
Since $A[j] \leq x$, i will be incremented and then $A[i]$ and $A[j]$ will be swapped.

(e)

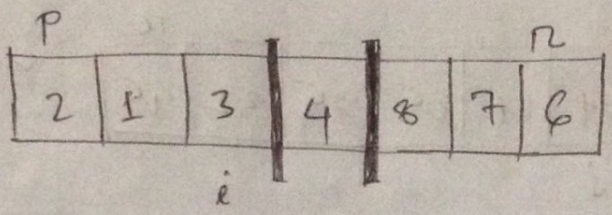


Since $A[j] \leq x$, i will be incremented and then $A[i]$ & $A[j]$ will be swapped.

(f)

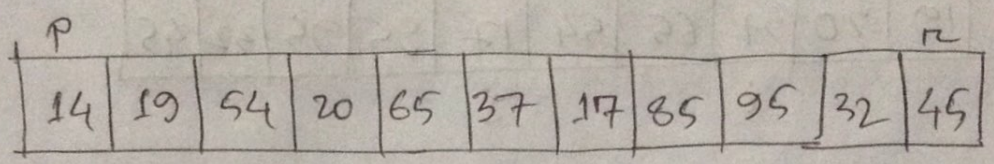


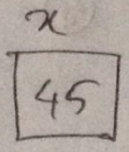
Now $A[i+1]$ and $A[r]$ will be exchanged and $(i+1)$ will be returned (i.e. 4).



(ii)

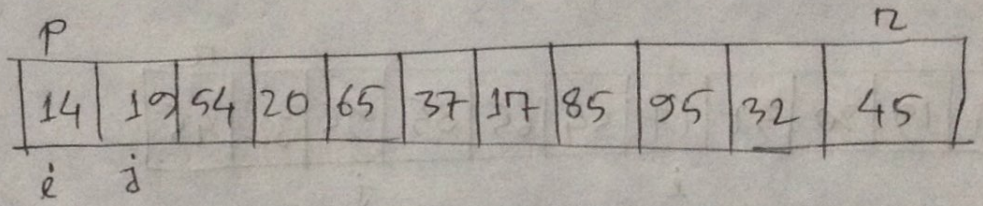
(a)



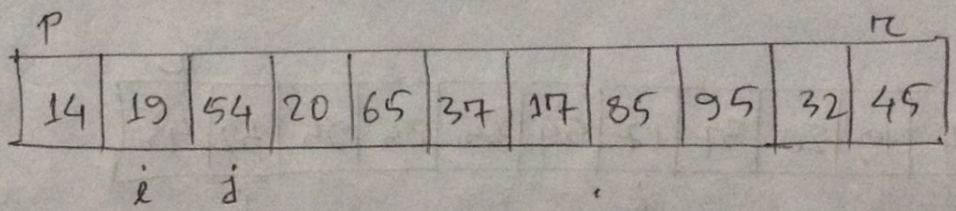


Since $A[j] \leq x$, i will be incremented and then A[i] and A[j] will be swapped. Here element 14 will be exchanged with itself.

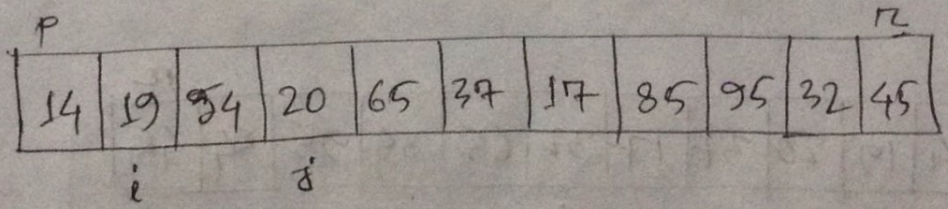
(b)



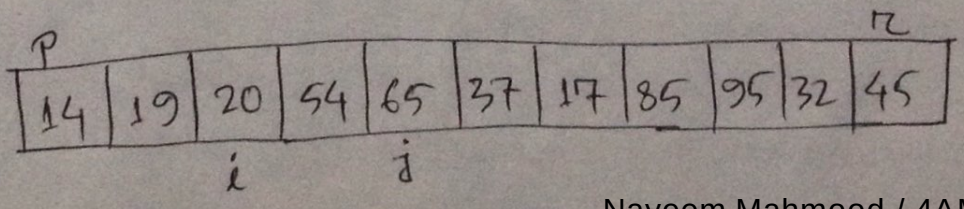
(c)



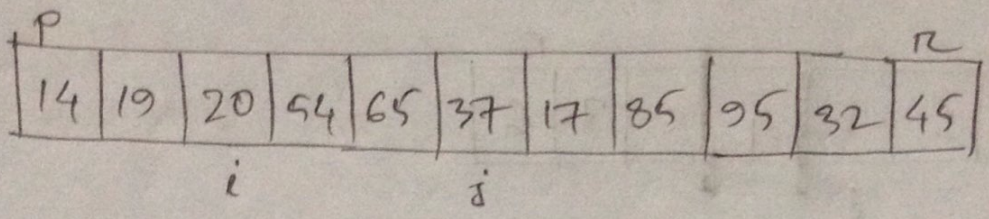
(d)



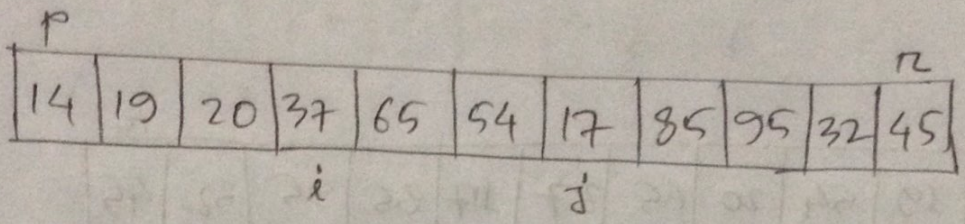
(e)



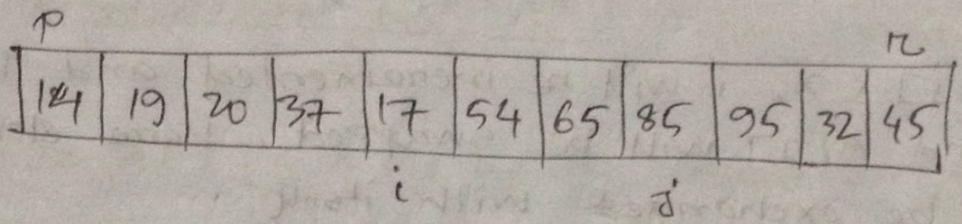
(f)



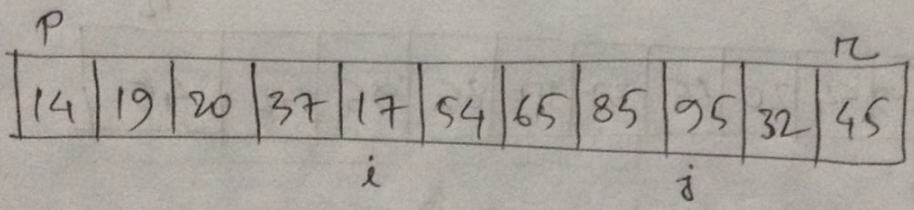
(g)



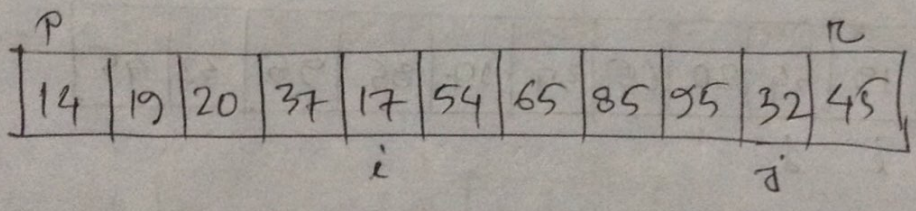
(h)



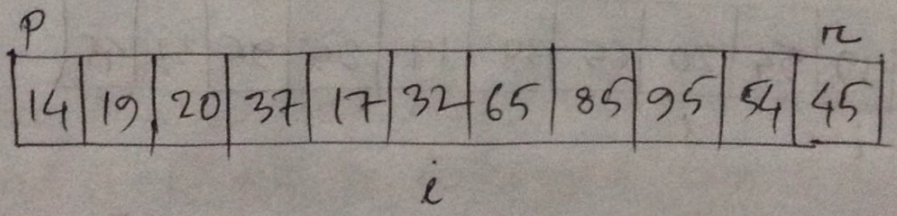
(i)

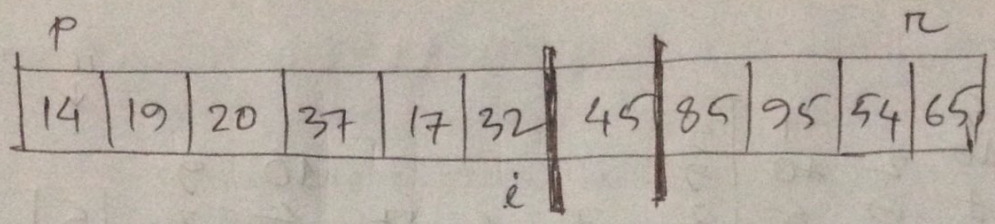


(j)



(k)





Consider the following array $A[p..r] = \langle 5, 4, 1, 9, 5, 2, 7, 6, 7, 4, 3, 10 \rangle$. Array index begins at $p = 5$ and ends at $r = 16$. Assume pivot $x = A[r]$.

i) What value will be returned by Partition procedure if run on array A?

ii) What value will be returned if $A[p]$ is exchanged with $A[r]$ before calling Partition?

iii) Let $A[p..r] = \langle 2, 4, 1, X, X, 8, X, 6, 7, X, 3, 5 \rangle$ where x are some unknown values. What is the maximum and minimum value that could be returned by Partition?

(i)

p r

5 4 1 9 5 2 7 6 7 4 3 10

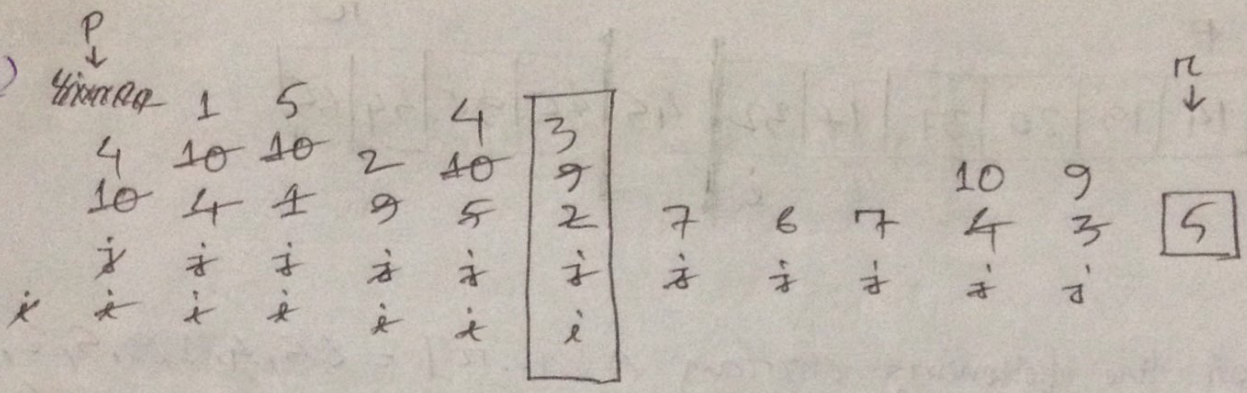
i i i i i i i i i i i

i i i i i i i i i i

i i i i i i i i i i

$i + 1 = 16$ will be returned.

(ii)

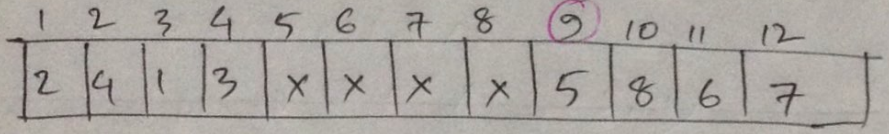


$i+1 = 11$ will be returned.

(iii) Here Pivot element $= A[r] = 5$

The maximum value could be returned if all the unknown values are less than or equal to 5 and the value could be 9.

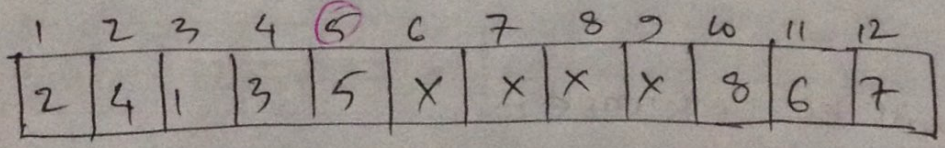
[*** Just to understand, not to write in script :



]

The minimum value could be returned if all the unknown values are greater that 5 and the value could be 5.

[***



]

Question: Suppose we are sorting an array of eight integers using Quick Sort algorithm and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Which statement is correct?

1. The pivot could be either 7 or the 9.
2. The pivot could be the 7 but it's not the 9.
3. The pivot is not the 7, but it could be the 9.
4. Neither the 7 nor the 9 is the pivot.

Answer: Statement (1) is correct.

Explanation: All the numbers on the left side of 7 are smaller than 7 and all the numbers on the right side of 7 are greater than 7. This applies for 9 too.

So according to the definition of Pivot, either 7 or 9 could be the pivot.

Nayeem Mahmood / 4AM / C161026

Write down two advantages of Quick sort over Merge sort.

1. Auxiliary Space : Quick sort is an in-place sorting algorith which takes no additional storage space to perform sorting while Merge Sort algorithm takes extra storage.

2. Locality of Reference: Quick sort in particular exhibits good cache locality and this makes it faster than Merge-sort in many cases like in virtual memory environment.

Quicksort(A) // A is an array

1. if p < r
2. q = Partition(A, p, r)
3. Quicksort(A, p, q-1)
4. Quicksort(A, q+1, r)

What will happen if you delete line 1 (if p < r) from above algorithm? Explain

In simple terms, we can say stack overflow will happen.

Explanation: There will be such a situation when q will be 1. In that case, the third line might be executed as Quicksort(A, 1, 0) assuming p = 1. since the condition at line 1 is removed, again q will have value 1 and again Quicksort(A, 1, 0)

Nayeem Mahmood / 4AM / C161026

will be called. This chain of recursive callings will continue until the stack is overflown and eventually a Runtime Error is will be followed.

田 Write down the recurrence equation of the Quicksort algorithm in which Partition always produces a 9-to-1 proportional split.

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn$$

田 Why do we use randomized version of Quicksort? You may use an example to clarify your point.

Both the deterministic and randomized quicksort algorithms have the same best case running times of $O(n \log n)$ and the same worst case running times of $O(n^2)$. The difference is that with the deterministic algorithm, a particular input can elicit the worst case behavior. With the randomized version, however, no input can also always elicit the worst case behavior. The reason it matters is that, depending on how partitioning is implemented, an input that is already sorted or almost sorted, can elicit the worst case behavior in deterministic quick sort. That's why we use the randomized version of quick sort.

MERGE SORT

Algorithm

```
MERGE (A, P, q, r)
1.  n1 = q - P + 1
2.  n2 = r - q
3.  Let L[1.. n1+1] and R[1.. n2+1] be new arrays
4.  for i = 1 to n1
5.      L[i] = A[P+i-1]
6.  for j = 1 to n2
7.      R[j] = A[q+j]
8.  L[n1+1] = ∞
9.  R[n2+1] = ∞
10. i = 1
11. j = 1
12. for k = P to r
13.     if L[i] ≤ R[j]
14.         A[k] = L[i]
15.         i = i+1
16.     else A[k] = R[j]
17.         j = j+1.
```

Nayeem Mahmood / 4AM / C161026

```
MERGE-SORT (A,P,r)
1. if p < r
2.     q = ⌊(p+r)/2⌋
3.     MERGE-SORT (A,p,q)
4.     MERGE-SORT (A,q+1,r)
5.     MERGE (A,p,q,r)
```

Consider an array for merge sort :

$A = \langle 5, 2, 3, 1, 6, 1, 7 \rangle$

i) How many Merge-Sort call will be there when the initial call is on array A ?

ii) How many Merge operation will be there ?

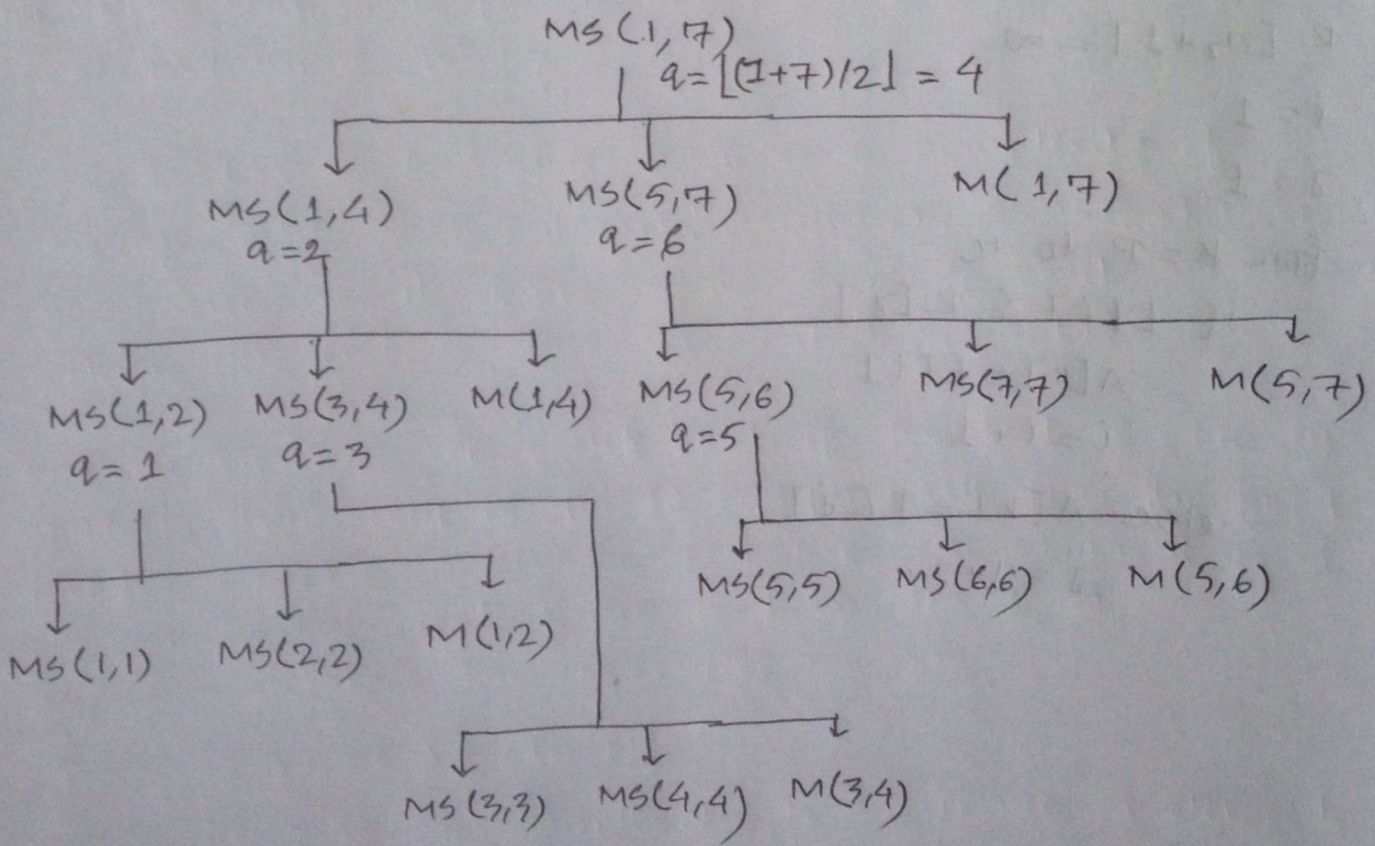


Fig: Recursive call tree for given array

Nayeem Mahmood / 4AM / C161026

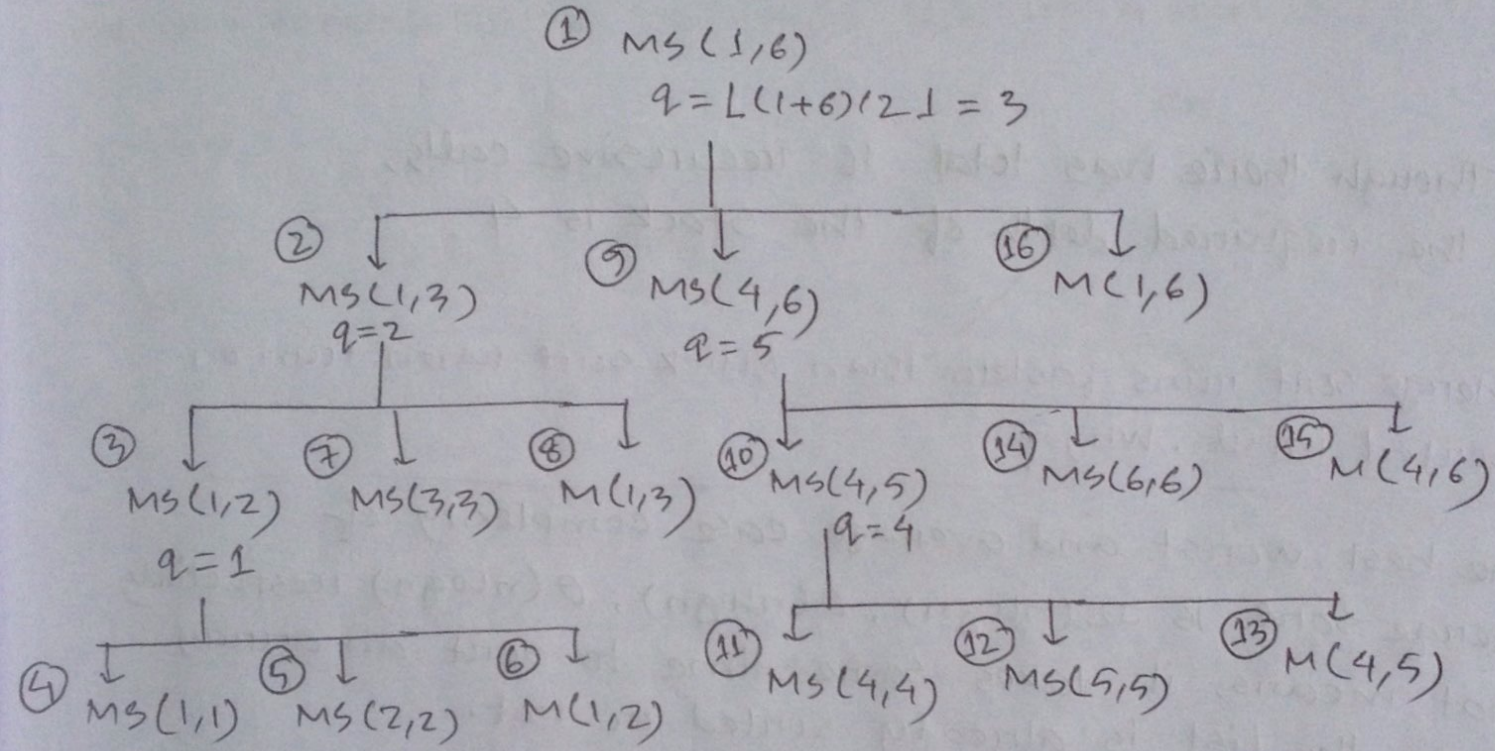
i) Total 13 calls including MS(1,7)

ii) 6 Merge operation will be there.

Consider A = <8, 3, 1, 9, 5, 2> be an array for merge sort.

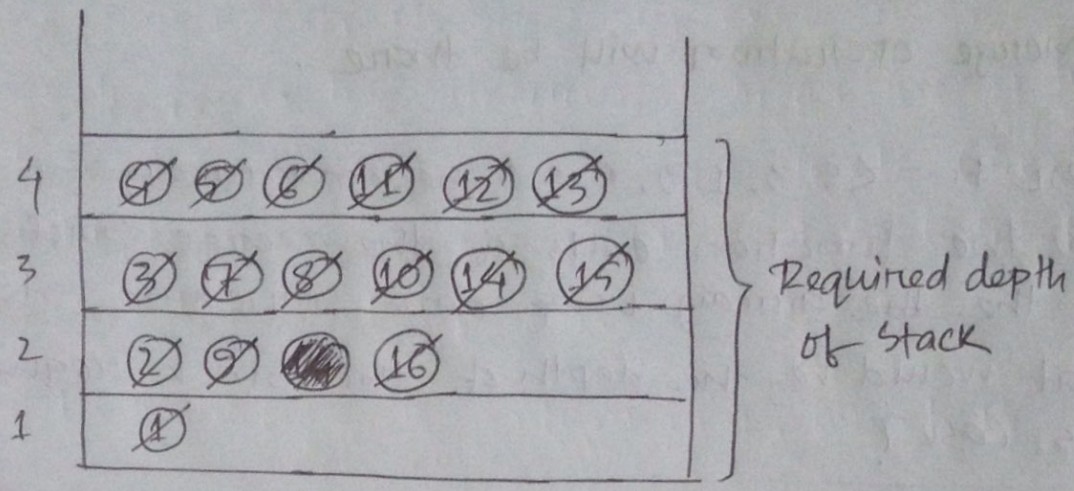
i) Show the function calls of the merge sort algorithm for the ~~the~~ array using data structure stack.

ii) What would be the depth of the stack required for this case?



The numberings inside circles are showing the order of recursive calls.

i)



ii) Though there was total 16 recursive calls, the required depth of the stack is 4.

Merge sort runs faster than Quick sort when run on sorted input. Why?

The best, worst and average case complexity of Merge sort is $\Omega(n \log n)$, $O(n \log n)$, $\theta(n \log n)$ respectively. That means, it takes same time to sort an array either the list is already sorted or not.

But in the case of Quick sort, if the list is already sorted, there will be $(n-1)$ comparisons for the first pivot, $(n-2)$ comparisons for the second pivot and so on. ~~In general, there will be total~~

We can express the recurrence as

$$T(n) = O(n) + T(n-1)$$

where $O(n)$ is the complexity to compare n elements.
We can deduce that,

$$T(n) = O(n^2)$$

Nayeem Mahmood / 4AM / C161026

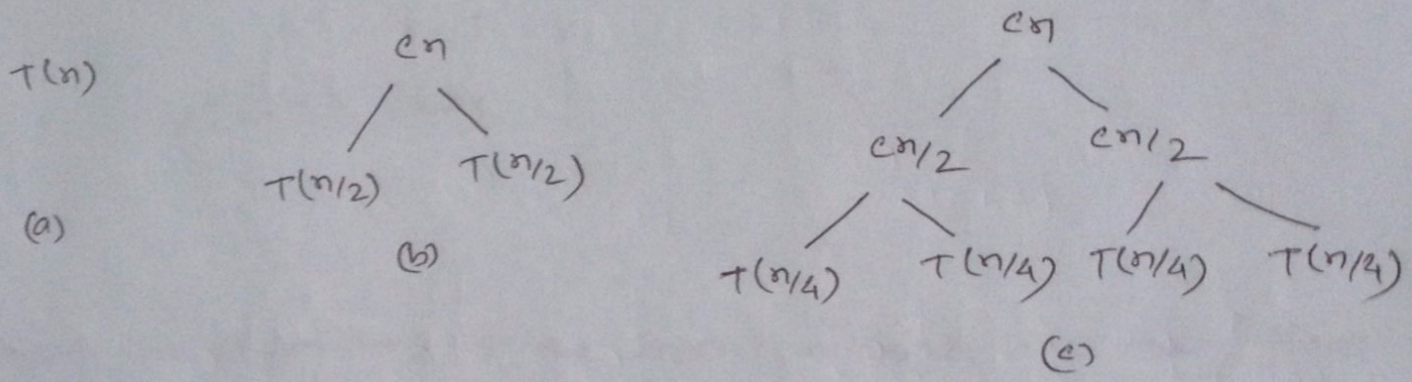
which is the worst case behavior of Quick sort algorithm

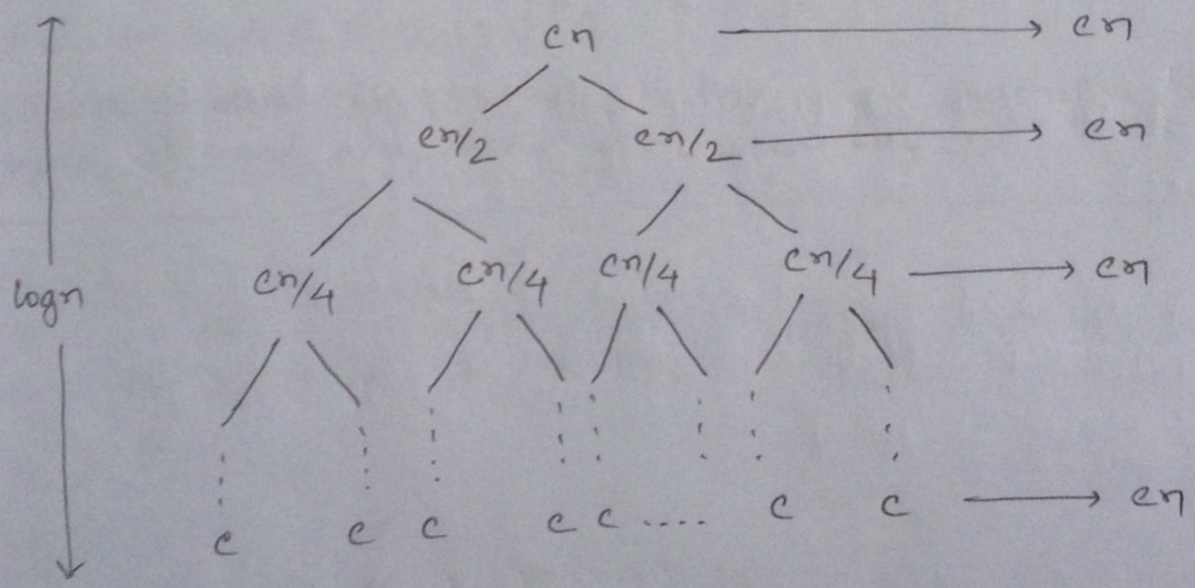
⊞ Construct a recursion tree for the worst case running time of the merge sort. Find out its total cost in term of θ notation.

The given recurrence equation is

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

We can construct the recursion tree in 4 steps:





(d)

Step (a) shows $T(n)$ which progressively expands in steps (b) - (d) to form the recursion tree. The fully expanded tree in step (d) has $\log n + 1$ levels (i.e., it has height $\log n$, as indicated), and each level contributes a total cost of cn.

The total cost, therefore is $cn\log n + cn$, which is $\Theta(n\log n)$.

Nayeem Mahmood / 4AM / C161026

INSERTION SORT

Algorithm

Insertion-Sort (A) // A is an array

```
1.  for j=2 to A.length
2.      key = A[j]
3.      i=j-1
4.      while (i > 0) and (A[i] > key)
5.          A[i+1] = A[i]
6.          i = i-1
7.      A[i+1] = key
```

Given the algorithm of the Insertion Sort, how many times Line 4 will be executed if run on the array $A = \langle 15, 2, 4, 6, 7, 9, 11 \rangle$?

Where should you change in the given algorithm if you want to sort only first four elements ?

---

j=2

	15	2	4	6	7	9	11	Key: 2
②		i	j					

j=3

	2	15	4	6	7	9	11	key: 4
②		i	j					

$j=4$

			15					key
(2)	2	4	15	~~6~~	7	9	11	6
		i	i					

$j=5$

				15				key
(2)	2	4	6	15	~~7~~	9	11	7
			i	i				

$j=6$

					15			key
(2)	2	4	6	7	15	~~9~~	11	9
				i	i			

$j=7$

						15		key
(2)	2	4	6	7	9	15	~~11~~	11
					i	i		

So, the statement at line 4 will be executed for 12 times.

If I want to sort only the first 4 elements, I should re-write statement at line 1 as

for $i=2$ to 4

[4] Given the algorithm of Insertion sort,

i) Why does the value of j at line 1 starts from 2?

ii) What would be the result if you rewrite line 4 as:

while(A[i] > key) ?

iii) How many comparisons (i.e. A[i] > key at line 4) will be there while you apply insersion sort on array A = <5, 1, 2, -3>

iv) Calculate the running time of Insertion sort algorithm for the worst case.

---

i) A list (or sub-list) containing one element is ~~alaw~~ always sorted. Thats why, j starts from 2.

ii) If we omit the expression (i > 0), lower bound of i will never be checked which will cause invalid array index accessing i.e. indexes less than 1 will be accessed; theoretically which don't exist but practically, key will be compared to some garbage values and the original list might be altered with those unexpected garbage values.

iii)

j=2

5 1 2 -3 key [1]

i j

[5] > [1] ✓

j = 3

		5		Key
1	5	2	-3	2
i	j			

5 > 2 ✓

1 > 2 ×

j = 4

	1	2	5	Key
1	2	5	-3	-3
i	j	j	j	

5 > -3 ✓

2 > -3 ✓

1 > -3 ✓

So, in total 6 comparison will take place.

iv)

Statements	Cost	Times
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{j=2}^{n} (t_j)$
5	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$n-1$

Total running time,

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 (n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \left(\frac{n(n+1)}{2} - 1\right) +$$

$$c_5 \left(\frac{n(n-1)}{2}\right) + c_6 \left(\frac{n(n-1)}{2}\right) + c_7 (n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n^2}{2} + \frac{n}{2} - 1\right)$$

$$+ c_5\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_6\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_7(n-1)$$

$$= [illegible]$$

$$\therefore T(n) = \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n$$

$$- (c_2 + c_4 + c_5 + c_7)$$

We can express this worst case running time as $an^2 + bn + c$ for constants a, b, c that again depend on the statement costs c_i ; it is thus a quadratic function of n.

⊞ Let an array consists of the following elements :

1, 5, 10, 12, 15, 29

i) Is the arrangement defined as the "best case" for insertion sort ?

ii) Is it the best case for quick sort ?

---

i) As the list is already sorted, it can be defined as the best-case for insertion sort.

ii) As the list is already sorted, it can't be the best but the worst case for quick sort.

[4] Assume that an array A contains 10000 numbers in sorted order. You are asked to sort them using Insertion-sort, Merge-sort and Quick-sort. Which algorithm will take minimum time and which will take maximum time? Justify your answer?

Insertion-sort algorithm will take minimum time. Because for each $j = 2, 3, \ldots, n$, we can find that $A[i] \leq key$ where i has its initial value of $j-1$. So, there will be just one comparison operation for 1 number and we obtain the best case complexity as $\Omega(n)$

Quick-sort algorithm will take maximum time. Since list is sorted, every pivot is in its proper place. However, comparisons and exchanging operations will take place for all pivots and we obtain the worst case complexity as $O(n^2)$.

In case of Merge-sort, any type of input takes the same amount of time and the average case complexity that we get is $\theta(n \log n)$.

When is Insertion-sort a good choice for sorting an array?

i) Each component of the array requires a large amount of memory.

ii) Each component of the array requires a small amount of memory.

iii) The array has only a few items out of place.

iv) The processor speed is fast.

True statement is (iii). Insertion sort works better in sorted or almost sorted list.

Nayeem Mahmood / 4AM / C161026

Selection Sort

The selection sort algorithm is given below:

```
Selection-Sort(A) // A is an array
1.  for j = 1 to A.length - 1
2.      iMin = j
3.      for i = j+1 to A.length
4.          if A[i] < A[iMin]
5.              iMin = i
6.      if (iMin != j)
7.          exchange A[j] with A[iMin]
```

i) Find how many times each line will execute for the input $A = \langle 15, 2, 4, 6, 7, 9, 11 \rangle$?

ii) Express the time it takes on the input A in Θ-notation.

iii) Intuitively tell what will be the worst case and best case running time.

---

i)

Statements	Times
1	7
2	6
3	$\sum_{K=2}^{7} K$
4	$\sum_{K=2}^{7} K - 1$
5	$\sum_{K=2}^{7} K - 1$

Statement	Times
6	6
7	6

This is not an exact calculation but approximate.

ii)

Statement	Times	Cost
1	n	c_1
2	$n-1$	c_2
3	$\sum_{k=2}^{n} k$	c_3
4	$\sum_{k=2}^{n} k - 1$	c_4
5	$\sum_{k=2}^{n} k - 1$	c_5
6	$n-1$	c_6
7	$n-1$	c_7

Total cost,

$$T(n) = c_1 n + c_2(n-1) + c_3 \sum_{k=2}^{n} k + c_4\left(\sum_{k=2}^{n} k - 1\right) + c_5\left(\sum_{k=2}^{n} k - 1\right)$$

$$+ c_6(n-1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3\left(\frac{n(n+1)}{2} - 1\right) + c_4\left(\frac{n(n-1)}{2}\right)$$

$$+ c_5\left(\frac{n(n-1)}{2}\right) + c_6(n-1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + c_4\left(\frac{n^2}{2} - \frac{n}{2}\right)$$

$$+ c_5\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_6(n-1) + c_7(n-1)$$

$$= \left(\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right)n^2 + \left(c_1 + c_2 + \frac{c_3}{2} * \frac{c_4}{2} - \frac{c_5}{2} + c_6 + c_7\right)n$$

$$- (c_2 + c_3 + c_6 + c_7)$$

we can write this as

$$T(n) = an^2 + bn + c$$

$$= O(n^2)$$

Nayeem Mahmood / 4AM / C161026

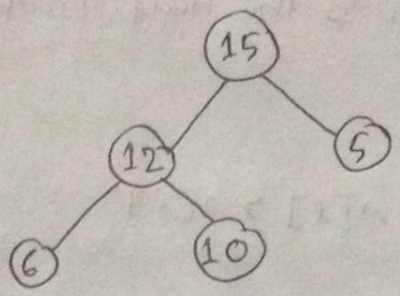
iii) The worst-case and best-case running times are $O(n^2)$ and $\Omega(n^2)$ respectively.

Nayeem Mahmood / 4AM / C161026

HEAP

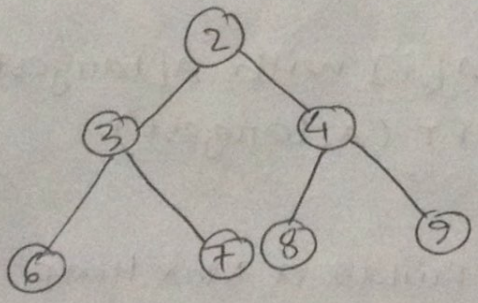
Max-Heap

The root contains the ~~maximum etc~~ largest element of the tree or sub-tree.



Min-Heap

The root contains the lowest element of the tree or sub-tree.



- If there are total n elements in a heap, the height of the heap, $h = \lfloor \log_2 n \rfloor$
- If "h" is the height of a heap, the maximum number of nodes $= 2^{h+1} - 1$
- In an n-element heap, there are $\lceil \frac{n}{2} \rceil$ nodes are leaf nodes and $n - \lceil \frac{n}{2} \rceil$ are non-leaf nodes.

Nayeem Mahmood / 4AM / C161026

Number of nodes at height "h" is at most

$$\left\lceil \frac{n}{2^{h+1}} \right\rceil$$

Related Algorithms

MAX-HEAPIFY (A, i) // Maintains the heap property

```
1.  l = 2i
2.  r = 2i + 1
3.  if l ≤ A.heap-size and A[l] > A[i]
4.          largest = l
5.  else
6.          largest = i
7.  if r ≤ A.heap-size and A[r] > A[largest]
8.          largest = r
9.  if largest ≠ i
10.         exchange A[i] with A[largest]
11.         MAX-HEAPIFY (A, largest)
```

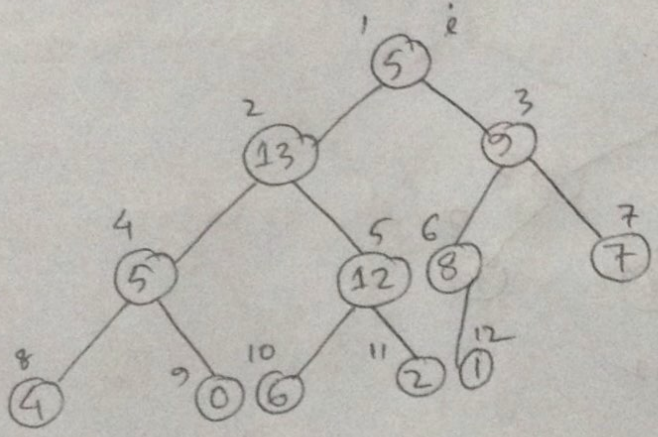
BUILD-MAX-HEAP (A) // Builds a Max Heap

```
1.  A.heap-size = A.length
2.  for i = ⌊A.length/2⌋ down to 1
3.          MAX-HEAPIFY (A, i)
```

HEAP-EXTRACT-MAX (A)

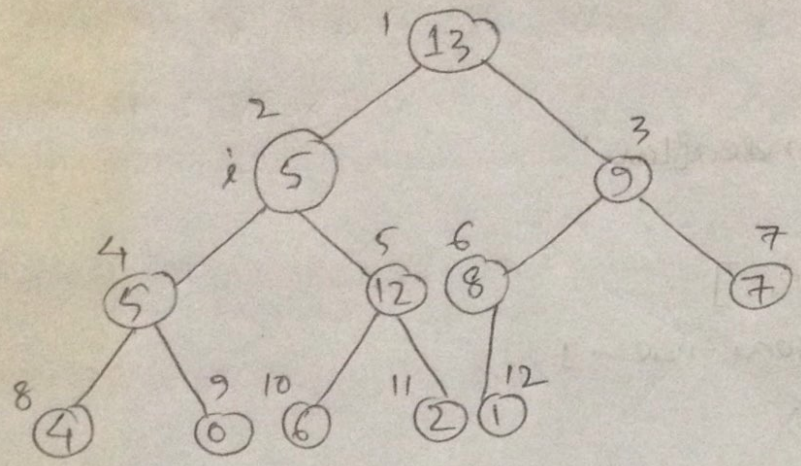
1. if A.heap-size < 1
2. error "heap underflow"
3. max = A[1]
4. A[1] = A[A.heap-size]
5. A.heap-size = A.heapsize - 1
6. MAX-HEAPIFY (A, 1)
7. return max

Illustrate the operation of MAX-HEAPIFY (A, 1) on the following array: A = <5, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1>

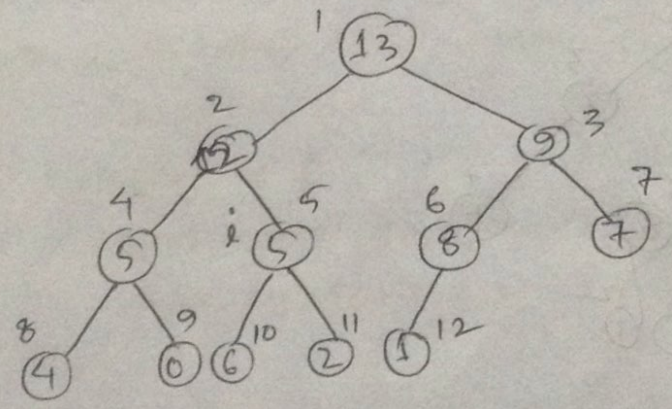


The initial configuration, with A[1] at node i = 1 is violating the max-heap property since it is not greater than both children. The max-heap property is restored for node 1 in figure below by exchanging A[1] with A[2].

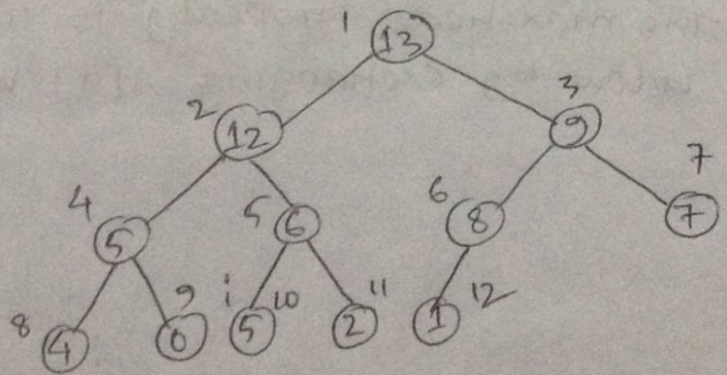
Nayeem Mahmood / 4AM / C161026



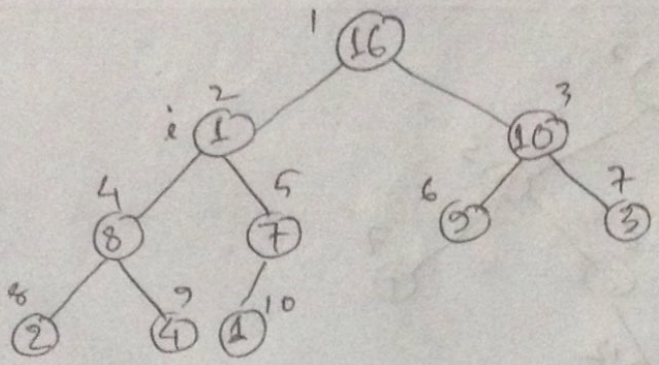
Now this configuration violates max-heap property for node 2. The recursive call MAX-HEAPIFY (A, 2) now has $i = 2$. To restore the property, A[2] and A[5] will be swapped.



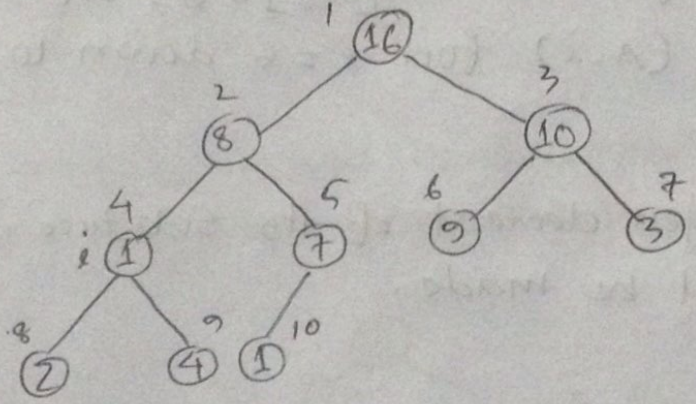
Max-heap property is violated again for node $i = 5$. So, A[5] will be swapped with A[10].



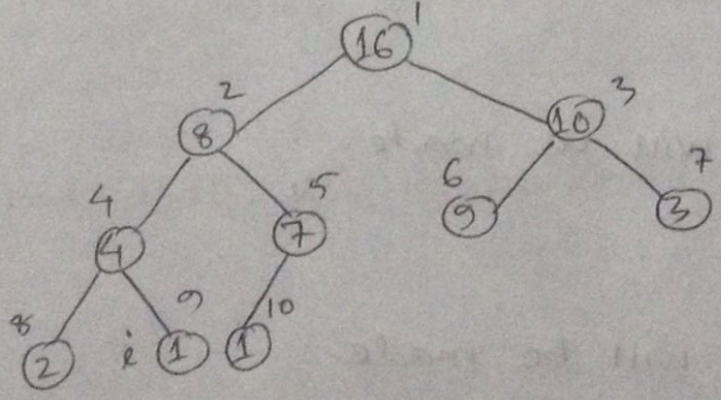
Illustrate the operation MAX-HEAPIFY(A, 2) on the array : $A = \langle 16, 1, 10, 8, 7, 9, 3, 2, 4, 1 \rangle$



A[2] violates the max-heap property, will be exchanged with largest child A[4].

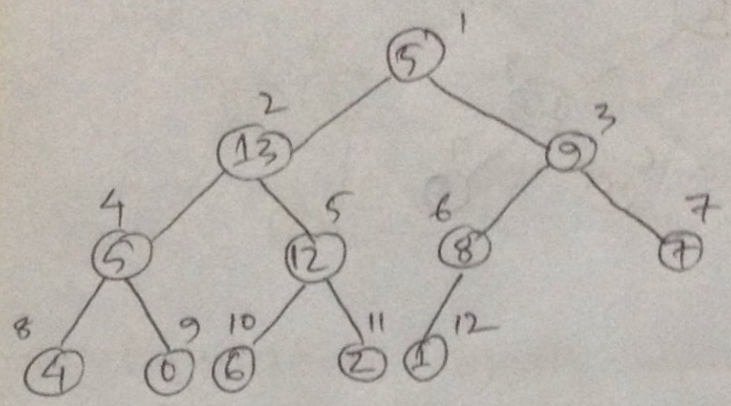


Again A[4] violates the max-heap property, will be exchanged with largest child A[9].



Nayeem Mahmood / 4AM / C161026

[1] Illustrate the Build-Max-Heap(A) on the following array: A=⟨5, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1⟩



Here, A.heap-size = A.length = 12. The non-leaf nodes starts from 1 to $\lfloor 12/2 \rfloor = 6$. We will call MAX-HEAPIFY(A, i) for i = 6 down to 1.

i = 6 →

A[6] is the largest element of its sub-tree, No changes will be made.

i = 5 →

A[5] is the largest element of its sub-tree. No changes will be made.

i = 4 →

No changes will be made.

i = 3 →

No changes will be made.

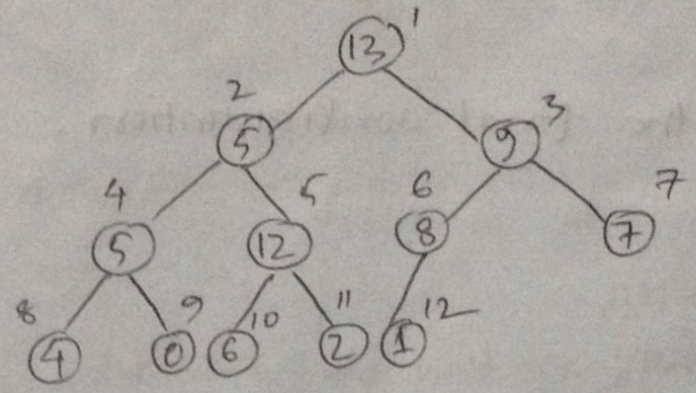
Nayeem Mahmood / 4AM / C161026

$i = 2$ →

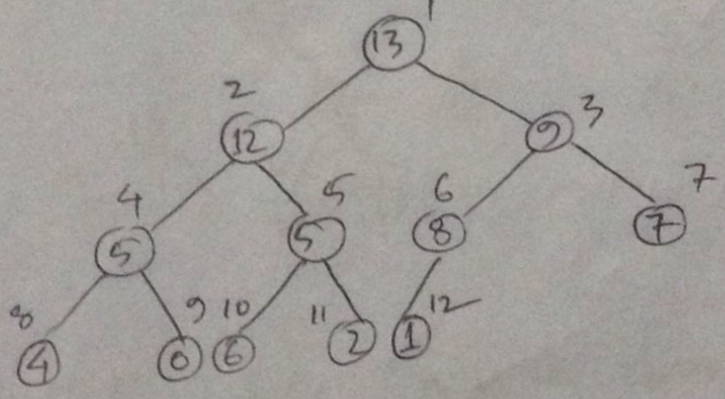
No changes will be made.

$i = 1$ →

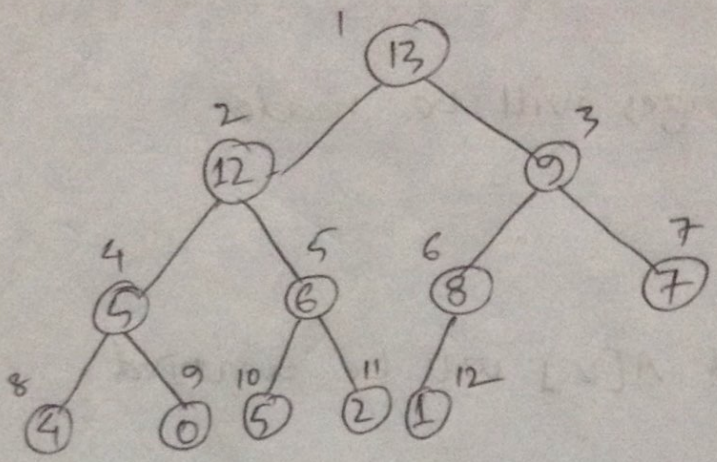
A[1] and A[2] will be swapped.



A[2] and A[5] will be swapped since A[2] violates the max-heap property.



A[5] and A[10] will be swapped.

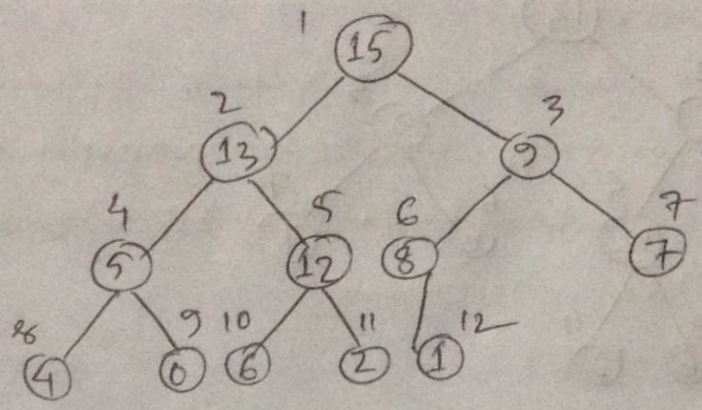


This is the final configuration.

Nayeem Mahmood / 4AM / C161026

Illustrate the operation of HEAP-EXTRACT-MAX on the heap

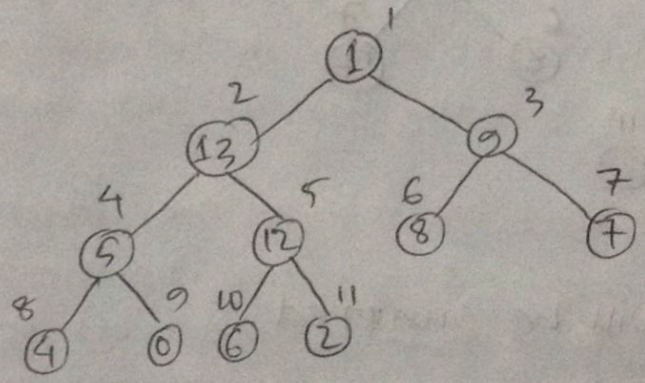
$A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$



Here, A.heap-size = 12

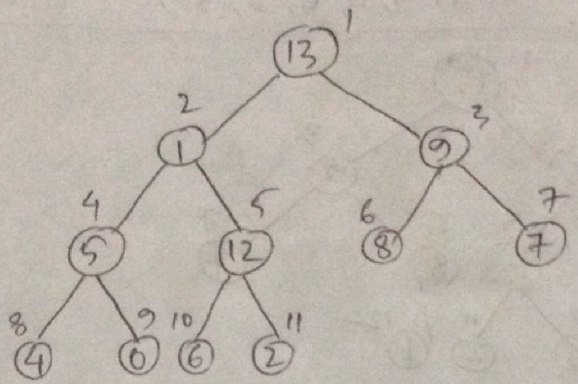
max = A[1] = 15

Now, A[1] ~~and A[12]~~ will be ~~swapped~~ replaced by A[12] and A.heap-size will become 12 - 1 = 11.

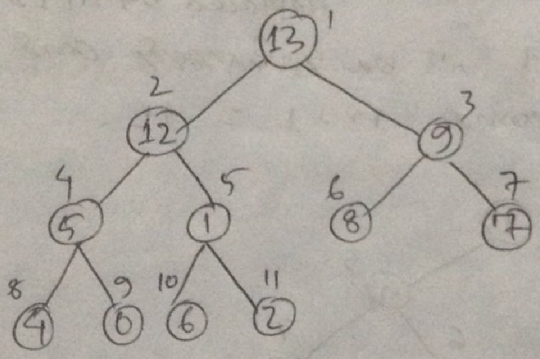


Now, MAX-HEAPIFY (A, 1) will be called.

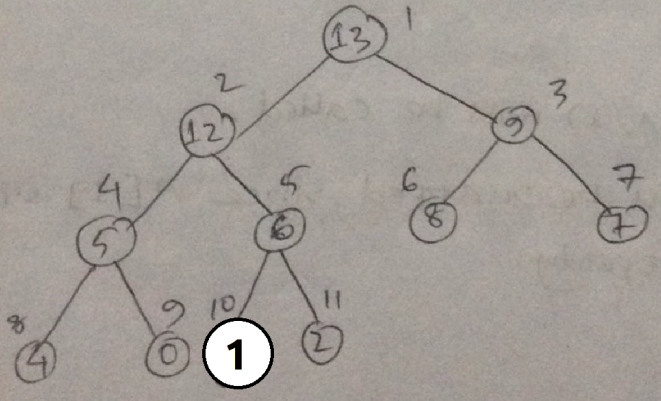
A[2] and A[1] will be swapped since A[1] violates the max-heap property.



A[2] and A[5] will be swapped.



A[5] and A[10] will be swapped.



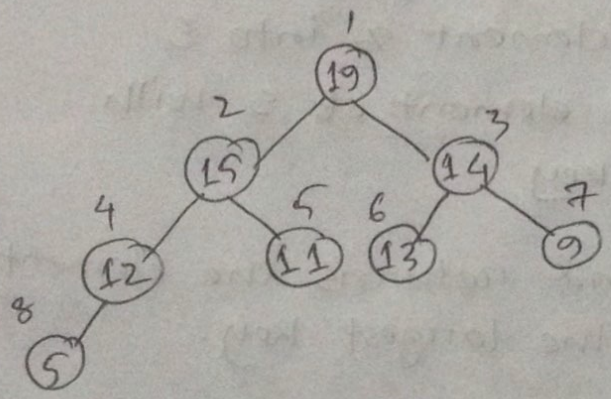
This is the final configuration of the given heap.
15 will be returned.

Suppose we have an array A where all the elements are same. Can it be considered as a Max-Heap? Can it be considered as a Min-Heap?

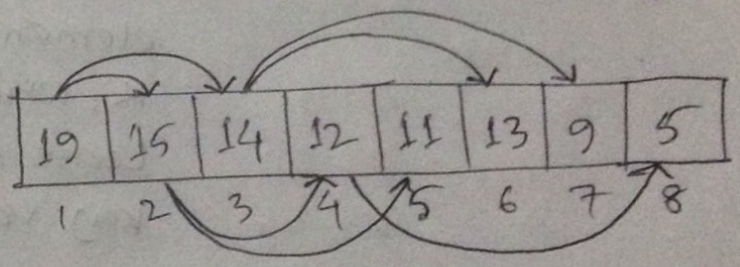
The given array A can be considered as both a Max-Heap and a Min-Heap.

With suitable example, show how a max-heap can be viewed as i) A binary tree ii) An array

---



(a)



(b)

Fig: (a) Tree view of Max-heap
(b) Array view of Max-heap

Nayeem Mahmood / 4AM / C161026

What are the operations supported by a priority queue?

A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key. A priority queue can be a 1) Max-Priority-Queue or 2) Min-Priority-Queue.

A max priority queue supports the following operations:

1) INSERT (S, x) : inserts the element x into S
2) MAXIMUM (S) : returns the element of S with the largest key
3) EXTRACT-MAX (S): removes and returns the element of S with the largest key.
4) INCREASE-KEY (S, x, K) : increases the value of element x's key to the new value K, which is assumed to be at least as large as x's current key value.

Apart from INSERT (S, x) operation, a min-priority queue supports MINIMUM (S), EXTRACT-MIN (S), DECREASE-KEY (S, x, K).

Nayeem Mahmood / 4AM / C161026

MASTER THEOREM

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

$a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number.

1) If $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

2) If $a = b^k$,

   i) If $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

   ii) If $p = -1$, then $T(n) = \theta(n^{\log_b a} \log\log n)$

   iii) If $p < -1$, then $T(n) = \theta(n^{\log_b a})$

3) If $a < b^k$,

   i) If $p \geq 0$, then $T(n) = \theta(n^k \log^p n)$

   ii) If $p < 0$, then $T(n) = \theta(n^k)$

Nayeem Mahmood / 4AM / C161026

Solve the recurrences using master method/ theorem :

i) $T(n) = 4T(n/2) + n^3$

ii) $T(n) = 5T(n/5) + n^2$

iii) $T(n) = 9T(n/3) + n$

iv) $T(n) = 2^n T(n/2) + n^n$

v) $T(n) = 2T(n/2) + n\log n$

vi) $T(n) = 2T(n/2) + \frac{n}{\log n}$

vii) $T(n) = 0.5T(n/2) + \frac{1}{n}$

viii) $T(n) = 54T(n/8) - n^2\log n$

ix) $T(n) = 2T(n/2) + \sqrt{n}$

x) $T(n) = 4T(n/2) + cn$

xi) $T(n) = 4T(n/16) + \sqrt{n}$

---

i) Given,

$$T(n) = 4T(n/2) + n^3$$

Here,

$a = 4,$

$b = 2,$

$k = 3,$

$P = 0$

$b^k = 2^3 = 8$, $\therefore a < b^k$.

Since $P > 0$, $T(n) = \theta(n^k \log^p n)$

$= \theta(n^3 \log^0 n)$

$= \theta(n^3)$

Nayeem Mahmood / 4AM / C161026

ii) Given,

$$T(n) = 5T(n/5) + n^2$$

Here,

$a = 5,$
$b = 5,$
$k = 2,$
$p = 0$

$b^k = 5^2 = 25$, $\therefore a < b^k$.

Since $p > 0$, $T(n) = \theta(n^k \log^p n)$

$= \theta(n^2 \log^0 n)$

$= \theta(n^2)$

iii) Given,

$$T(n) = 9T(n/3) + n$$

Here,

$a = 9,$
$b = 3,$
$k = 1,$
$p = 0$

$b^k = 3^1 = 3$, $\therefore a > b^k$

$\therefore T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_3 9})$

$= \theta(n^{\log_3 3^2})$

$= \theta(n^{2\log_3 3})$

$= \theta(n^2)$

Nayeem Mahmood / 4AM / C161026

iv) Given,

$$T(n) = 2^n T(n/2) + n^n$$

Comparing the given relation with,

$T(n) = aT(n/b) + \theta(n^k \log^p n)$, we get $a = 2^n$. But a must be any constant where $a \geq 1$. So, we can't apply Master Theorem to solve the relation.

v) Given,

$$T(n) = 2T(n/2) + n\log n$$

Here,

$a = 2$,
$b = 2$,
$k = 1$,
$p = 1$

$b^k = 2^1 = 2$, $\therefore a = b^k$

Since $p > -1$, $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_2 2} \log^{1+1} n)$

$= \theta(n \log^2 n)$

$= \theta(n \log\log n)$

vii) Given,

$$T(n) = 2T(n/2) + \frac{n}{\log n}$$

$$= 2T(n/2) + n\log^{-1} n$$

Here, $a = 2$,
$b = 2$,
$k = 1$,
$p = -1$

$b^k = 2^1 = 2$, $\therefore a = b^k$.

Since $p = -1$, $T(n) = \Theta(n^{\log_b a} \log\log n)$

$= \Theta(n^{\log_2 2} \log\log n)$

$= \Theta(n \log\log n)$

viii) Given, $T(n) = 54T(n/8) - n^2\log n$

Master Theorem can't be applied here due to negative term i.e. $-n^2\log n$.

Nayeem Mahmood / 4AM / C161026

ix) Given,

$$T(n) = 2T(n/2) + \sqrt{n}$$

Here,

$a = 2,$

$b = 2,$

$k = 1/2,$

$p = 0$

$b^k = 2^{1/2}$, $\therefore a > b^k$

$\therefore T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_2 2})$

$= \theta(n)$

x) Given,

$$T(n) = 4T(n/2) + cn$$

Here,

$a = 4,$

$b = 2,$

$k = 1,$

$p = 0.$

$b^k = 2^1 = 2$, $\therefore a > b^k$

So, $T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_2 4})$

$= \theta(n^{\log_2 2^2})$

$= \theta(n^{2\log_2 2})$

$= \theta(n^2)$

Nayeem Mahmood / 4AM / C161026

xi) Given,

$$T(n) = 4T(n/16) + \sqrt{n}$$

Here, $a = 4$,

$b = 16$,

$k = 1/2$,

$p = 0$

$b^k = 16^{1/2} = 4$, $\therefore a = b^k$.

Since $p = 0$ i.e. $p > -1$, $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_{16} 4} \log^{0+1} n)$

$= \theta(n^{\log_{16} \sqrt{16}} \log n)$

$= \theta(n^{\log_{16} 16^{1/2}} \log n)$

$= \theta(n^{1/2 \log_{16} 16} \log n)$

$= \theta(\sqrt{n} \log n)$

Nayeem Mahmood / 4AM / C161026