Binary Search, Matrix Multipli → 2D, Selection Sort → Algo

Date: _____

$2^3 = 8 \rightarrow \log_2(8) = 3$

$b^y = x \rightarrow \log_b(x) = y$

```
int x = 0          - 1
while (x<n)        - 1        | logn
{ n/=2             - 1+1      | 2(logn - 1)
  x+=1 }           - 1+1      | 2(logn - 1)
```

$5\log n - 3$

n=16

0 < 16

1 < 8

2 < 4

3 < 2

$2^4 \rightarrow 16^x \rightarrow \log_2(16) = 4$

logn

```
for(int i=1; i<=n; i*=2)
{                 1   logn+2   logn+1
  Print(i) }
  logn+1
```

$3\log n + 5$

$1 \leq 16$

$2 \leq 16$

$4 \leq 16$

$8 \leq 16$

$16 \leq 16$

$32 \leq 16$

6

$2^4 \rightarrow 16^2$

logn + 2

```
int Method(int n)
{ int x=0;          1   | 1
  while(n>1)        1   | logn+1
  { n=n/2;          1+1 | logn·(2)
    x=x+1; }        2   | logn(2)
  return x;         1   | 1
}
```

n = 8

$5\log(n) + 5$

$O(\log(n))$

Asymptotic notation

Date: _____

$f(x)$ $30x + 8$

$g(x) = n$

$C = 31$

$30x + 8 \leq 31n$

$n = 1$

$278 \leq 279$ $n \div 8$ $n = 9$

$\boxed{n_0 = 9}$

$n \geq n_0$

$n \geq 8$ & $C = 31$

Types of Algorithms

Date: _____

Probablistic Algorithm: In this algo, chosen values are used in such a way that the probablity of chosen each value is known & controlled Eg: Randomize Quick Sort.

Heuristic Algorithm: This type of algo, is based largely on optimism & often with minimal theoretical support. Here error can't be controlled but estimated how large it is

Approximate Algorithm: is obtained this as precised as required in decimal notation. In other words it specifies the error we willing to accept.
Eg: two figure accuracy or 8 figures or whatever is required.

Algorithm Approches: •) Brute Force algo •) Greedy Algo •) Recursive Algo •) Backtracking Algo •) Divide & Conquer Algo •) Dynamic programming Algo •) Randomized Algo

Brute Force Algo: Hit & Try Algo, to achieve their goals.
Eg: For 4-digit pin algo chose digit 0-9, try all possible combinations one by one until get pin right.
↳ worst case it will take 10,000 tries.

Greedy Algo: the solution is built part by part - The decision to choose the next part is done on the basis that it gives an immediate benefit - It never consider the choices that had taken previously. Choosing best option at that time, not in long ru
Problem Solved: Dijkstra Shortest Path Algo, Prim's Algo, kruskal's Algo, Huffman Coding etc.

Recursive Algo: Based on recursion, solved by breaking it into subproblems of t same type & calling own self again & again until the problem is solved wit the help of a base condition.
Such common problem that is solved using recursive algorithms are Facto of Number, Fibonacci Series, Tower of Hanoi, DFA for Graph etc

Date: _____

Divide & Conquer Algorithm: the idea is to solve the problem in two section the first section divides the problem into subproblems of the same type. The second section is to solve the smaller problem independently & then add the combined result to produce the final answer to the problem. Some common problem solved are: Binary Search, Merge Sort, Quick Sort Matrix Multiplication etc.

Dynamic Programming Algorithm: also known as memoization technique cuz in this the idea is to store the previously calculated result to avoid calculating it again & again - In Dynamic Programming, divide the complex problem into smaller overlapping subproblem & store results for future use. Problems solved are: knapsack, Weighted Job Scheduling, Floyd Warshall etc.

Randomized Algo: we use random number, it helps to decide the expected outcome - Decision to choose random number so it gives immediate benefit. Common Problem: Quicksort: we use random number for selecting the pivot.

Algorithm Specifications: Input, Output, Definiteness, Finiteness & Correctness.

Date: _____

Space Complexity:

```
int fun(int[2n] A, int 2 n)
{ int sum=0
  for(int i=0; i<n; i++)
      sum += A[i]
  return Sum;
```

2n+8 O(n)

```
Float[] fun(Float[4n] A; Float[4n] B; int 2 n){
  Float[4n] C;
  for(int i=0; i<n; i++)
    C[i] = A[i] + B[i]
  return C;
```

16n+4 O(n)

Time Complexity:

```
int fun(int a, int b)
   int C = a+b
   return C;
```

3 units

```
for(int i=n; i>2; i--)
   print
```

n=5

5>2
4>2
3>2
2>2

i=n	1	1
i>2	1	n-1
i--	1	n-2
Print	1	n-2

3n-4 O(n)

```
int Sum=0;
for(int i=0; i<5; i++)
    Sum = Sum+i;
Print Sum;
```

Sum	1	1	1
i=0	1	1	1
i<5	1	6	6
i++	1	5	5
Sum=Sum+i	2	5	10
Print	1	1	1
			24

24 units O(1)

```
For(int i=0; i<n; i++)
    print(i);
```

i=0	1	1
i<n	1	n+1
i++	1	n
print	1	n
		3n+2

3n+2 O(n)

Date: _____

```
for(int i=0; i<=n; i++)
    print(i);
for (int j=1; j<=m; j++)
    print(j)
```

	if nested loop		n=3
i=0	1	1	0≤3
i≤n	1	n+2	1≤3
i++	1	n+1	2≤3
Print(i)	1	n+1	3≤3
j=1	1	1 (n+1) → n+1	4≤3
j≤m	1	m+1 (n+1) → nm+m+n+1	m=3
j++	1	m (n+1) → mn+m	1≤3
P(j)	1	m (n+1) → mn+m	2≤3
			3≤3
			4≤3

$3n+3m+7$ $O(n+m)$ $O(mn)$ $3mn+5n+3m+7$

```
for(int i=n; i>1; i--)
{ for(int j=0; j<n; j++)
  { while(k<n)
    { k++ }}}
```

			For i
i=n	1	1	n=3 3>1
i>1	1	n	2>1
i--	1	n-1	1>1
			For j
j=0	1	1 (n-1) → n-1	0<3
j<n	1	(n+1)(n-1) → n^2-1	1<3
j++	1	n(n-1) → n^2-n	2<3
			3<3
			For k
k<n	1	n+1 (n)(n-1) → n^3-n^2	k=0 0<3
k++	1	n (n)(n-1) → n^2-2n+1	1<3
			2<3
			3<3

$3n^3+n^2+n-2$ $O(n^3)$

```
for (int i=0; i<n; i++)
{ for (j=1; j<n; j++)
  { print }}
```

$3n^2+n+2$ $O(n^2)$

$1 < \log(n) \overset{\sqrt{n}}{<} n < n\log(n) < n^2 < n^3 < 2^n \ldots < n^n$

Asymptotic Notations: Date: _____

Asymptotic Analysis: To compare 2 algorithms with running times $f(n)$ and $g(n)$
we need rough measures that characterizes how fast each function grows.
↳ use rate of growth.

Rate of growth: Buying elephant & goldfish
Cost: cost_of_elephant + cost_of_goldfish
Cost ~ cost_of_elephant (approximation) Consider elephant cuz of higher polynomial.

Types of Analysis:
Worst Case: Provide upperbound on running time. [Algo would not run longer, no matter what input are: Guarantee.
Best Case: Provide lowerbound on running time. Input is the one for which algo runs the fastest.

Lowerbound ≤ RunningTime ≤ UpperBound.

AverageCase: Provides a prediction about running time. Assume that input is random.

O notation: asymtotic "less than" Ω notation: asymtotic "greater than" Θ notation: asymptotic "equality"
$f(n) = Og(n)$ implies: $f(n)$ "≤" $g(n)$ $fn = \Omega(g(n))$ implies: $f(n)$ "≥" $g(n)$ $f(n) = \Theta(g(n))$ implies $f(n)$=

Eg: $f(n) = 2n^2 + 4$ this is the case
$g(n) = n^2$ it is true for Ω, O c is constant

$n \rightarrow$ shows on the last intersection of two fn. After this $n \cdot cg(n)$ will alway grow
This condition will be when we attach c with $g(n)$. $n \geq n_0$
↱ The function $f(n) = O(g(n))$ such that $f(n) \leq c * g(n)$, $n \geq n_0$
Big O Notation: $f_A(n) = 30n + 8 \rightarrow O(n)$ $f_B(n) = n^2 + 1 \rightarrow O(n^2)$
$O(n^2)$ is fast-growing than $O(n)$
$n^4 + 100n^2 + 10n + 50 \rightarrow O(n^4)$ $10n^3 + 2n^2 \rightarrow O(n^3)$ $n^3 - n^2 \rightarrow O(n^3)$
constant
10 is $O(1)$ 1273 is $O(1)$

Polynomial different ho aur right side wala high ho tou true hoga.

•) if g(n) is not given then take the value from f(n) with highest polynomial

Date: _____

Calculation for Big O:

$f(n) = 2n^2 + n$, $g(n) = n^2$, $c = 3$

$f(n) \leq c * g(n)$

$2n^2 + n \leq 3(n^2)$

$n \leq n^2 \Rightarrow 1 \leq 1$

$\therefore n \geq n_0$

$n \geq 1$

$n_0 = 1$

$f(n) = 30n + 8$, $g(n) = n$, $c = 31$

$f(n) \leq c \cdot g(n)$

$30n + 8 \leq 31n$

$8 \leq n$

$n_0 = 8$

$30(8) + 8 \leq 31(8)$, $30(9) + 8 \leq 31(9)$

$248 \leq 248$ $278 \leq 279$

Q) $2n^2 = O(n^3)$

$2n^2 \leq cn^3$

$c = 1$ & $n_0 = 2$

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

$n \leq cn^2$

$c = 1$ & $n_0 = 1$

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

$n^2 \leq cn^2$

$c = 1$ & $n_0 = 1$

Q) $n^2 = 1000n^2 + 1000n$, $c = 1001$

$1001n^2 = 1000n^2 + 1000n$

$n^2 = 1000n$

$c = 1001$ & $n_0 = 1000$

Ω notation → $f(n) \geq g(n)$ for all $n \geq n_0$

Θ notation → when polynomial of f(n) & g(n) is same

$f(n) = \Omega(g(n))$; $f(n) \geq c \cdot g(n)$

$f(n) = 2n^2 + n$, $2n^2 + n \geq c \cdot g(n^2)$

$2n^2 + n \geq 2n^2$ $n \geq 2n^2 - 2n^2$

$n \geq 0$ $\therefore n_0 = 0$ & $c = 2$

Q) Find upper bound of running time of quadratic func

$f(n) = 3n^2 + 2n + 4$

$f(n) \leq c \cdot g(n)$

$3n^2 + 2n + 4 \leq 3n^2 + 2n^2 + 4n^2$

$3n^2 + 2n + 4 \leq 9n^2$ $\therefore c = 9$

$1 \leq n$

$n_0 = 1$ & $c = 9$

Q) Find the lower bound of running time of quadratic functio

$f(n) = 3n^2 + 2n + 4$, $f(n) \geq c \times g(n)$

$3n^2 + 2n + 4 \geq 3n^2 \rightarrow$ true, for all $n \geq 1$

$3n^2 + 2n + 4 \geq n^2 \rightarrow$ true, for all $n \geq 1$

Above both inequalities are true & there exis such infinite inequalities.

So, $f(n) = \Omega g(n) = \Omega(n^2)$ for $c = 3$, $n_0 = 1$

$f(n) = \Omega(g(n)) = \Omega(n^2)$ for $c = 3$, $n_0 = 1$.

$5n^2 = \Omega(n)$, $cn \leq 5n^2 \Rightarrow c = 1$ & $n_0 = 1$

$n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Lect: 3
Last slides.

TSP & knapsack best known example of so called NP-Hard Problem.

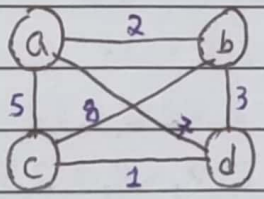
NP-Hard: algo whose time complexity solution doesn't fall in polynomial.

Date: _____

Exhaustive Search: is simply a brute-force approach to combinational problems.

Travelling Salesman Problem: Finding the shortest path by visiting all the vertices of graph atleast once - Hamiltonian-Circuit-Cycle that passes through all the vertices of graph exactly once, only first vertic visited again at the end.

Time Complexity: $(n-1)!$ ·) n is no. of vertices.

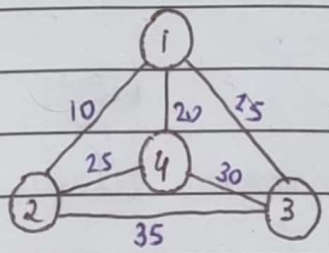


$a \xrightarrow{2} b \xrightarrow{8} c \xrightarrow{1} d \xrightarrow{7} a = 18$; $a-c-d-b-a = 11$

$a \xrightarrow{2} b \xrightarrow{3} d \xrightarrow{1} c \xrightarrow{5} a = 11$; $a-d-c-b-a = 18$

$a-c-b-d-a = 23$; $a-d-b-c-a = 23$

$(n-1)! \Rightarrow (4-1)! \Rightarrow 3! \Rightarrow \boxed{6}$ → Total possible combinations.



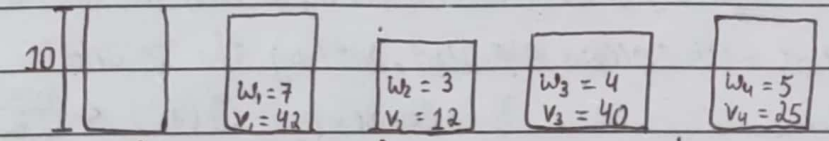
$1-2-4-3-1 = 80$; $1-3-4-2-1 = 80$

$1-4-3-2-1 = 95$; $1-2-3-4-1 = 95$

$1-4-2-3-1 = 88$; $1-3-2-4-1 = 88$

TC: 2^n

Knapsack Problem: Most valuable but less in weight. Example: Shoes theif Aeroplane



2^n → No. of sets

$2^4 = 16$

Subset	Total Weight	Total Value	Subset	Total Weight	Total Valu
$\emptyset$	0	0	{2,3}	7	52
{1}	7	42	{2,4}	8	37
{2}	3	12	{3,4}	9	65
{3}	4	40	{1,2,3}	14	not feasib
{4}	5	25	{1,2,4}	15	" "
{1,2}	10	54	{1,3,4}	16	" "
{1,3}	11	not feasible	{2,3,4}	12	" "
{1,4}	12	" "	{1,2,3,4}	19	" "

grows exponentially or faster.

Date: _____

Assignment Problem: Assigning n jobs to n persons (one person per job)
Check all the combinations that which person performs all the jobs in less time.
Time Complexity: $n! \rightarrow 4! \Rightarrow 24$

	Job1	Job2	Job3	Job4
Person1	9	2	7	8
Person2	6	4	3	7
Person3	5	8	1	8
Person4	7	6	9	4

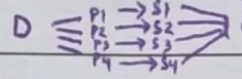
Rows → P1 P2 P3 P4 (Column) 3rd row, 4th column.

<1,2,3,4> → 9+4+1+4 = 18
<1,2,4,3> → 9+4+8+9 = 30
<1,3,2,4> → 9+3+8+4 = 24
<1,3,4,2> → 9+3+8+6 = 26
<1,4,3,2> → 9+7+1+6 = 23

<1,4,2,3>, <2,1,3,4>, <2,1,4,3>, <2,3,1,4>, <2,3,4,1>, <2,4,1,3>, <2,4,3,1>
<3,1,2,4>, <3,1,4,2>, <3,2,1,4>, <3,2,4,1>, <3,4,1,2>, <3,4,2,1>, <4,1,2,3>
<4,1,3,2>, <4,2,1,3>, <4,2,3,1>, <4,3,1,2>, <4,3,2,1>.

Bubble & Selection sort are Bruteforce approach.

Divide & Conquer: Divide a big problem in small sub problems. Time Complexity: nlogn



Selection Sort: We start selection sort by scanning the entire given list to find its smalle element & exchange it with the first element, putting the smallest element in its final positi in the sorted list - Then we scan the list, starting with second element, to find the smalle among the last n-1 elements & exchange it with second element, putting the second smallest element in its final position.

Time Complexity: $O(n^2) \Leftarrow \frac{n(n-1)}{2}$
Actually → $\Theta(n^2)$

Find min & swap.

for i ← 0 to n-2 do
  min ← i
  For j ← i+1 to n-1 do
    if A[j] < A[min]
      min ← j
  swap A[i] and A[min]

Swap with the minimum, place it to first.

43 9 26 4 15
min → min, min

4 9 26 43 15
i & min

4 9 26 43 15
swap, min → min

4 9 15 43 26
sorted, swap

4 9 15 26 43

Date: _____

Bubble Sort: Another brute-force application to sorting problem is to compare adjacent elements of the list and exchange them if they are out of order - By doing it repeatedly, we end up "bubbling up" the largest element to the last position on list. The next pass bubble up the second largest element and so on, until after n-1 pass list is sorted.

Fix any "out-of-order" number locally i.e,

↳ if two numbers are in decreasing order, swap them.

$O(n^2)$

6 5 3 1 8 7 2 4
5 6 3 1 8 7 2 4
5 3 6 1 8 7 2 4
5 3 1 6 8 7 2 4
5 3 1 6 7 8 2 4
5 3 1 6 7 2 8 4
5 3 1 6 7 2 4 8 } By this we got largest number in end. Now continue doing this.
3 1 5 6 2 4 7 8
1 3 5 2 4 6 7 8
1 3 2 4 5 6 7 8
1 2 3 4 5 6 7 8 → Sorted.

```
for i ← 0 to n-2 do
    for j ← 0 to n-2-i do
        if A[j+1] < A[j]
        swap A[j] and A[j+1]
```

Insertion-Sort(A)

```
For j = 2 to A.length
    key = A[j]
    i = j-1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key.
```

40	20	60	10	50	30

20 40 60
10 20 40 60
10 20 40 50 60

10	20	30	40	50	60

Sorte

insertion sort → n^2

Date: _____

Divide & Conquer Approach:

Divide the problem into subproblems, Conquer the problems by solving them recursively. Combine the solutions to the sub problems into the final solution.

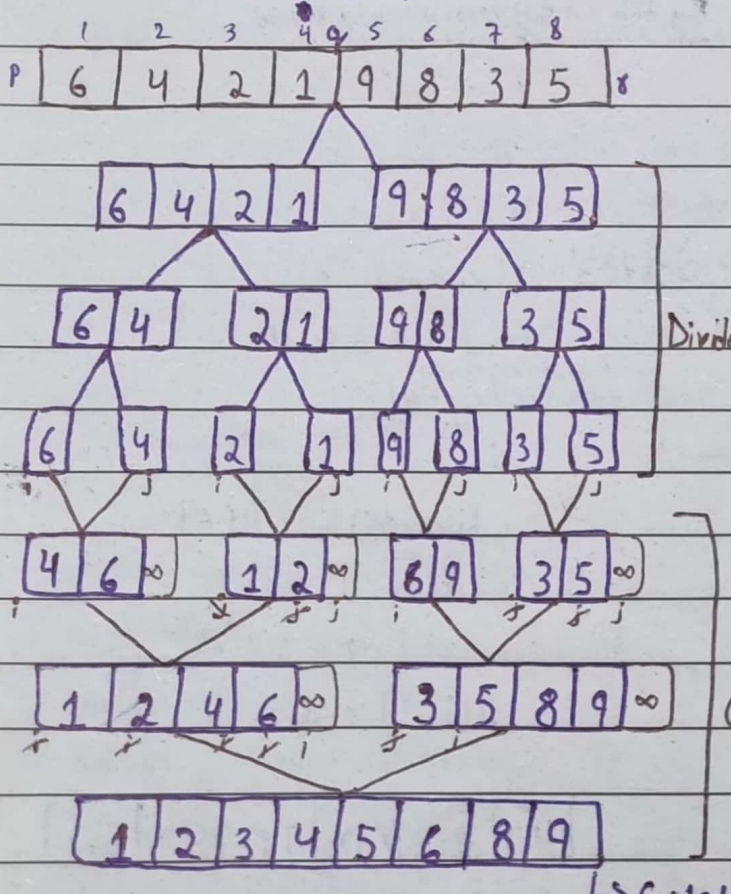
Merge Sort: Divide: divide the n element input into two subproblems. Time Complexity: nlogn

Conquer: sort the two sequences recursively.

Combine: merge the two sorted subsequences.

```
Merge-Sort(A,p,r)
  if p<r
    then q ← [(p+r)/2]
      Merge-Sort(A,p,q)
      Merge-Sort(A,q+1,r)
      Merge(A,p,q,r)
```

```
Merge(A,p,q,r)
  n1 = q-p+1
  n2 = r-q
  let L[1...n1+1] and R[1...n2+1] be new array
  For i=1 to n1
      L[i] = A[p+i-1]
  For j=1 to n2
      R[j] = A[q+j]
  L[n1+1] = ∞
  R[n2+1] = ∞
  i=1 , j=1
  For k=p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i+1
      else A[k] = R[j]
          j = j+1
```

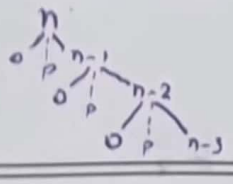


Divide

Conquer/Merge.

↳ Sorted Array.

Worst Case: (10) 20 30 40 50 60 70 +∞
P ⋯ ⋯ ⋯ ⋯ ⋯ ⋯ Q



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

$O(n^2) \rightarrow TC$

(DAC) Date: _____

Quick-Sort: Best case: O(nlogn), WorstCase $O(n^2)$, On Avg, it faster than heapsort due to highly optimized inner loop.
↳ divide a list into two sub-lists.

Steps: Pick an element, called pivot, from the list. → Left subset
Reorder the list so that all elements which are less than the pivot come before the pivot and all elements greater than the pivot come after it (equal value can go either way). → Right Subset
After this partitioning, the pivot is in its final position - Called partition operation
Recursively sort the sub-list of lesser elements & the sub-list of greater elements.
The base case of the recursion are lists of size zero or one, which are always sorted

↦ O(n)

```
Quicksort(A,p,r)
  if p<r
    q ← Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
```

```
Partition(A,p,r)
i ← p-1
for j ← p to r-1          → pivot
    if A[j] ≤ A[r]
        i ← i+1
        Swap A[i] and A[j]
Swap A[i+1] and A[r]
return i+1.
```

← P move to right until it get greater value than pivot
less than equal ← Q move to left until it get less value than pivot
+∞ we write so its greater than our pivot to stop P.
↓ Pivot

(35) 50 15 25 80 20 90 45 +∞
P ⟷ swap ⟷ Q ⋯ ∅

(35) 20 15 25 80 50 90 45 +∞
Swap P ⋯ Q ⋯ P ⋯ ∅

25 20 15 (35) 80 50 90 45 +∞

↳ 1st Pass : Divided Problem in 2 parts.

(25) 20 15 +∞ ; (80) 50 90 45 +∞
P Q P ; P P Q

15 20 25 35 (80) 50 45 90 +∞
RQ RP

15 20 25 35 • 45 50 80 90 Sorted

if P & Q are on same position or cross eachother then we replace Q with pivot.

Normal Case Reoccurrence: $2T(n/2) + n$
↳ When array is inversly sorted.
↳ When array is sorted

Quick sort is terrible: $O(n^2)$
Quick sort usually is: O(nlogn)
↳ if we always picked pivot that exactly cuts the array in half.
↳ Best case
Avg case

Median of 3 — the first, the last, the middle.

Another way to avoid worst case?
Inject randomness into the data.
↳ Still we could get unlucky and pick 'bad' partition at every step → $O(n^2)$

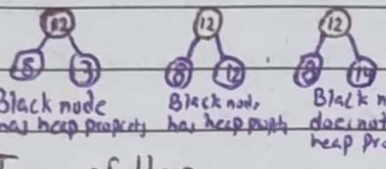
Quicksort is the fastest known sorting Algo
It is not stable
Optimum efficiency, the pivot must be choosen carefully.

Read → Tree
Types of tree

Quick Sort is generally faster but heapsort is better in time-critical applications. Date: _____

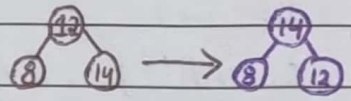
] HeapSort: Like merge sort, its running time is O(nlogn) - It sorts in place
↳ can be visualized as complete binary tree

Heap Condition: key in each node is larger (or equal to) child keys.



•) All leaf nodes automatically have the heap property.
•) A binary tree is a heap if all nodes in it have the heap property

Types of Heap:
A max-heap has the property that for every node i other than the root $A[Parent(i)] \geq A[i]$
A min-heap has the property that for every node i other than the root $A[Parent(i)] \leq A[i]$
Three main procedures: •) Heapify: to maintain the heap property.
•) Built-Heap: produces a heap from an unsorted input array
•) Heap-Sort: sorts an array in place.
Extract MAX and INSERT procedures, allow the heap data structure to be used a pirority queue.

Heapify:  Notice that the child may have lost the heap property.

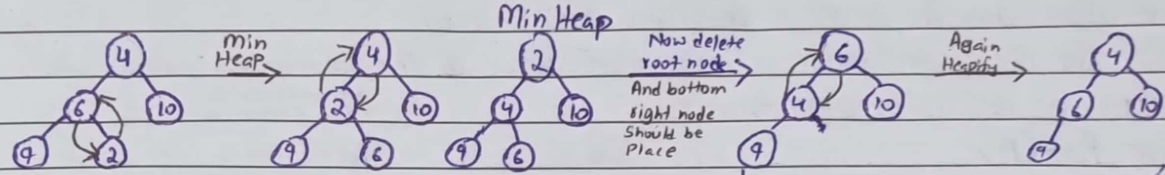
```
Max-Heapify(A,i)
  l = Left(i)
  r = Right(i)
  if l ≤ A.heap-size and A[l] > A[i]
      largest = l
  else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
      largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    Max-Heapify(A, largest)
    O(logn)
```

```
Build Heap(A)
heap-size(A) ← length(A)
for i ← [length[A]/2] down to 1
    do Heapify(A,i)
    O(nlogn)
```

```
HeapSort(A)
  Build Heap(A)
for i ← length[A] down to 2
    do exchange A[1] ↔ A[i
    heap size[A] ← heapsize[A]-1
    Heapify(A, 1).
    O(nlogn)
```

Date: _____

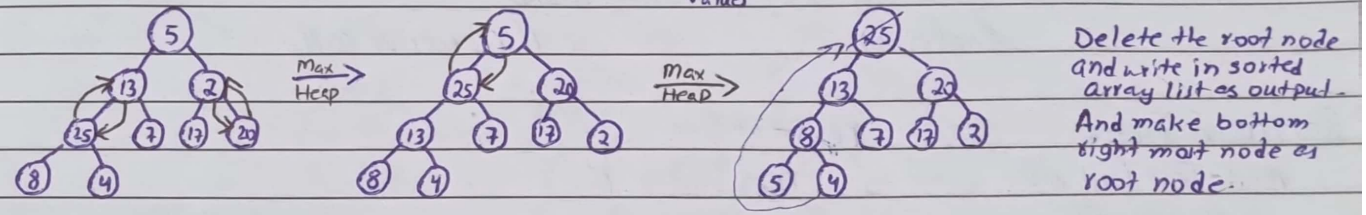
4 6 10 9 2



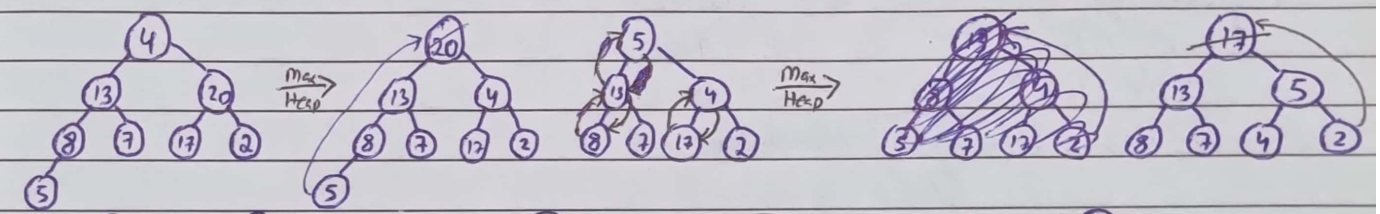
Deleted Elements: [2 4 6 9 10] Sorted

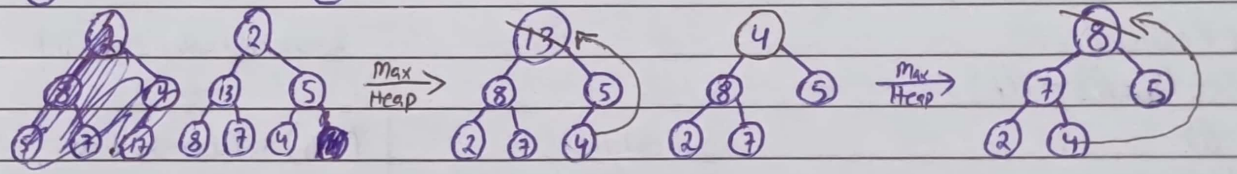
Q) A = [5, 13, 2, 25, 7, 17, 20, 8, 4] Sorted → [25, 20, 17, 13, 8, 7, 5, 4, 2]

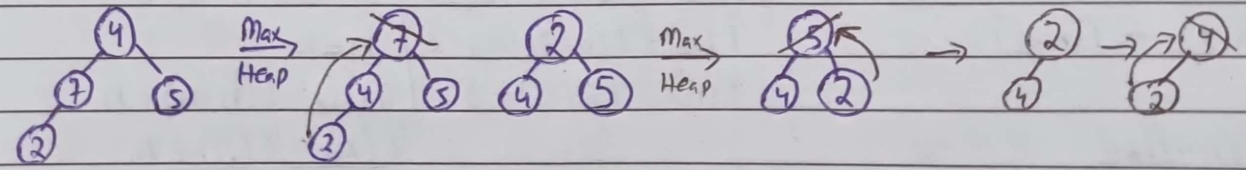
Deleted from the tree Values



Delete the root node and write in sorted array list as output. And make bottom right most node as root node.







Date: _____

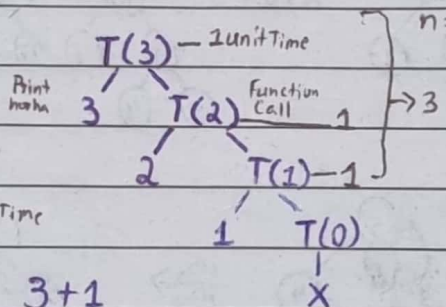
Recursive Algorithm: It is an algorithm which invokes itself.
A recursive algorithm looks at a problem backward.
↳ the solution for the current value n is a modification of the solution for previous values (n-1)

Basic Structure of Recursive algorithm:
- Head Recursion.
- Tail Recursion.

```
procedure foo(n)
    if n satisfy some condition then // base case
        return (possibly a value)
    else (possibly do some processing)
        return (possibly some processing using)
                foo(n-1)                    // recursive call.
```

Recursive Tree:

```
void Test (int n)
{ if (n>0)
  { print (n); — 1 unit Time
    Test(n-1); }}
```



3+1

$T(n) = n+1$ → Complexity

$O(n)$

Recursive Recurrance:

3 → k denotes constant

```
void Test (int n) — T(n)
{ if (n>0)
  { Print (n); — 1
    Test(n-1); — T(n-1)
  }}
```

$T(n) = T(n-1)+1$ if $n>0$

$T(n) = 1$ if $n=0$
↳ cuz if condition run

$T(n-1) = T(n-2)+1$

$T(n-1) = T(n-3)+1$

Substitute:

$T(n) = [T(n-2)+1]+1$

$T(n) [T(n-3)+1]+2$

$T(n) = T(n-3)+3$

$T(n) = T(n-k)+k$

assume $n-k=0$

$n=k$

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

$T(n) = T(0)+n$

$T(n) = n+1$

↳ $O(n)$

$0<3$
$1<3$
$2<3$
$3<3$] $n+1$

Date: _____

```
void Test(int n) — T(n)
{ if(n>0) — 1
  { for(int i=0; i<n; i++) — n+1
    { print(n); } — n
  Test(n-1) — T(n-1)
  }}
```

Recurrance:

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

$T(n) = T(n-1) + \boxed{2n+2}$ → neglect coofficient & constant we can write just n

$T(n) = T(n-1) + n$ $\quad \therefore n>0$

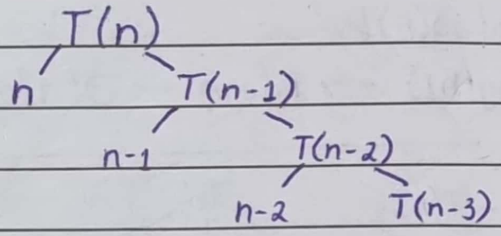
Substitute: $T(n) = [T(n-2) + (n-1)] + n$

$T(n) = T(n-2) + (n-1) + n$

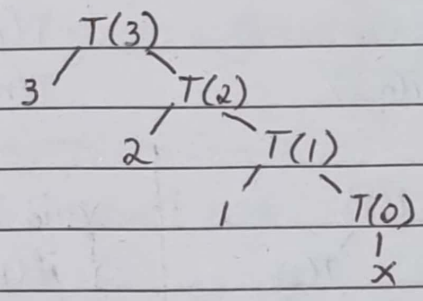
$T(n) = [T(n-3) + (n-2)] + (n-1) + n$

Using this $T(n) = T(n-1) + n$

Tree:



$T(N) = \frac{n(n+1)}{2} \rightarrow O(n^2)$



$0+1+2+\cdots+(n-2)+(n-1)+n = n(n+1)/2$

→ $3 \rightarrow k$

$T(n) = T(n-n) + (n-n+1) + (n-n+2) + \cdots + (n-1) + n$

$T(n) = T(0) + 1 + 2 + \cdots (n-1) + n$

$T(n) = 1 + (1 + 2 + \cdots (n-1) + n)$

$\boxed{T(n) = 1 + \frac{n(n+1)}{2}} \longrightarrow O(n^2)$

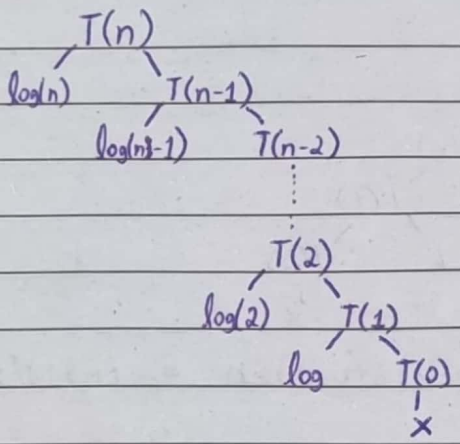
$n = 8$, 2^3

$1 < 8$
$2 < 8$
$4 < 8$
$8 < 8$

Date: _____

```
void Test(int n) — T(n)
{ if (n>0)
  { For (int i=0; i<n; i=i*2) — log(n)
    { print (n); } — log(n)
  Test(n-1); T(n-1)
}}
```

Tree:



$\log(n) + \log(n-1) + \dots + \log(2) + \log(1)$

$\therefore \log(a) + \log(b) = \log(ab)$

$\log(n*(n-1)*\dots*2*1)$

$\log(n!) \rightarrow n\log n \rightarrow O(n\log(n))$

Recurrance

$T(n) = T(n-1) + \log(n)$

Substitute:

$T(n) = [T(n-2) + \log(n-1)] + \log(n)$

$T(n) = [T(n-3) + \log(n-2)] + \log(n-1) + \log n$

$3 \rightarrow k$

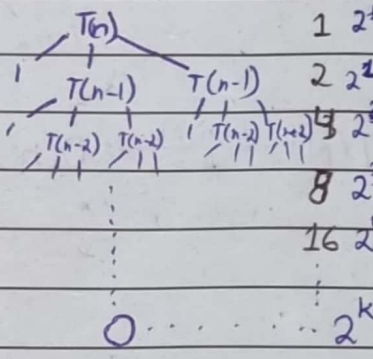
$T(n) = T(n-k) + \log(n-(k-1)) + \dots \log(n-1) + \log n$

$n - k = 0$

$T(n) = T(0) + \log(n)!$

$T(n) = 1 + \log(n!) \rightarrow n\log n \rightarrow O(n\log(n))$

```
void Test(int n) — T(n)
{ if (n>0)
  { print(n); — 1
  Test(n-1); — T(n-1)
  Test(n-1); — T(n-1)
}}
```



$1 + 2 + 2^2 + 2^3 \dots + 2^k$

$\therefore a + ar + ar^2 + ar^3 + \dots + ar^k = \frac{a(r^{k+1} - 1)}{r - 1}$

$a = 1, r = 2 = \frac{1(2^{k+1} - 1)}{2 - 1}$

$\hookrightarrow 2^{k+1} - 1$

$n - k = 0 \Rightarrow n = k$

$2^{n+1} - 1 \rightarrow O(2^n)$

Recurrence:

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

$T(n) = 2T(n-1) + 1 \quad n > 0$

$T(n) = 1 \quad n = 0$

$T(n-1) = 2T(n-2) + 1$

Substitute: $T(n) = 2[2T(n-2) + 1] + 1$

$T(n) = 2^2 T(n-2) + 2 + 1$

$T(n) = 2^3 T(n-3) + 2^2 + 2 + 1$

$n = k$

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots 2^2 + 2 + 1$

$\rightarrow T(n) = 2^n T(0) + 1 + 2 + 2^2 + \dots + 2^{n-1}$

$= \boxed{2^n + 2^{n-1}}$

$\hookrightarrow O(2^n)$

Date: _____

DFA: kinds of edges

DFA introduces an important distinctions among edges in the original graph.

Tree edge: encounters new (white) vertex.

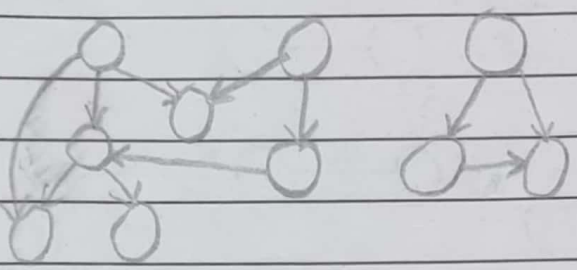
Back edge: from decendent to ancestor.

Forward edge: from ancestor to descendant.

Cross edge: between a tree or subroutine.

Note: Tree & back edges are imp; most algo don't distinguish forward & cross.

Directed Acyclic Graph: or DAG is directed graph with no directed cycles.



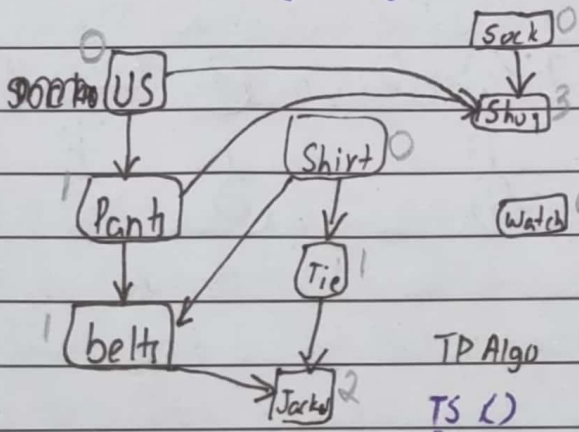
directed Graph G is acyclic iff a DFS of G yeilds no back edges.

→ Linear running TC.

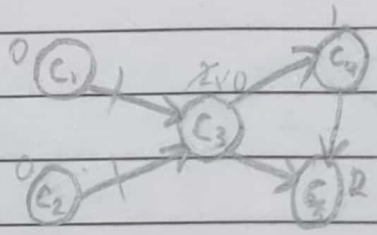
Topological Sort: of a DAG: Linear ordering of all vertices in graph G such that vertex u come before vertex v if edge $(u,v) \in G$.

Real World example: getting dressed:

Only work on directed & acyclic.



-) Only traverse those vertices whose indegree
-) Remove vertice whose indegree is 0



Watch, Sock, US, Pant, belt, Shirt, Tie, Jacket.

TP Algo

TS ()

{ Run DFS

When a vertex is finished, output it

Vertices are output in reverse topological order }

C_1, C_2, C_3, C_4, C_5

MIGHTY PAPER PRODUCT

Time: $O(V+E)$.

Spanning Tree. A connected subgraph 'S' of Graph G(V, E) is said to be spannin iff
↓
Tree is acyclic.
1) 'S' should contain all vertices of 'G'
2) 'S' should contain (|V| - 1) edges.

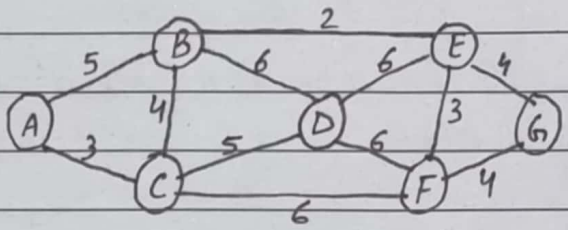
Find ST from Complete Graph
n^{n-2}

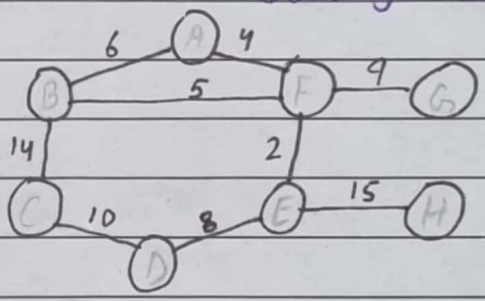
Date: _____

Greedy.

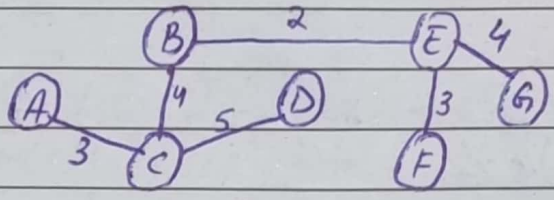
Minimum Spanning Tree (MST): For a weighted, connected, undirected graph, G, the total cost G is the sum of ~~weighted~~ weight on it edges.
A M-CST for G is MST of G that has least total cost.

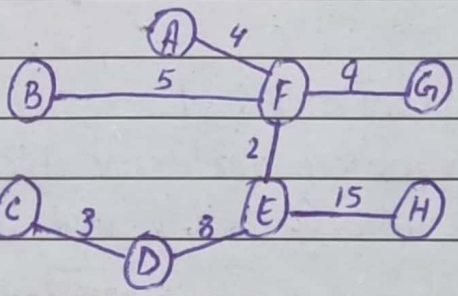
Kruskal's Algorithm: Undirected, Weighted
↳ Also work for disconnected cuz it doesn't need source.
-) When traversing make sure it shouldn't form cycle. -) Works on disjoint sets.
Running Time is O(ElogE) O(ElgV)
↳ Greater cuz edges are more than vertex
Edges: n-1
Initialization O(V)
Sorting: O(ElgE)

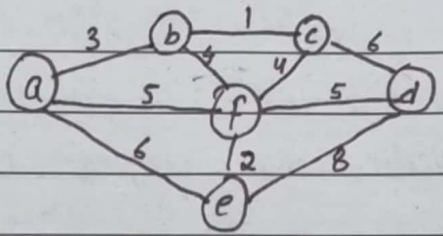


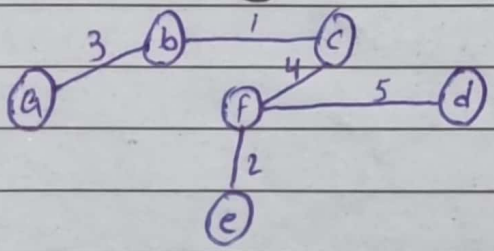


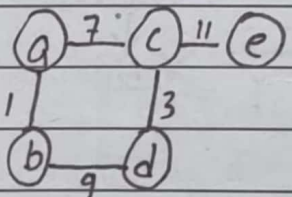
n-1
7-1=6

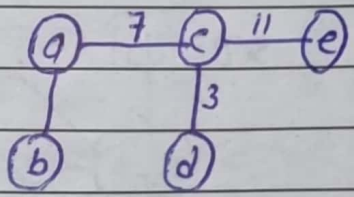










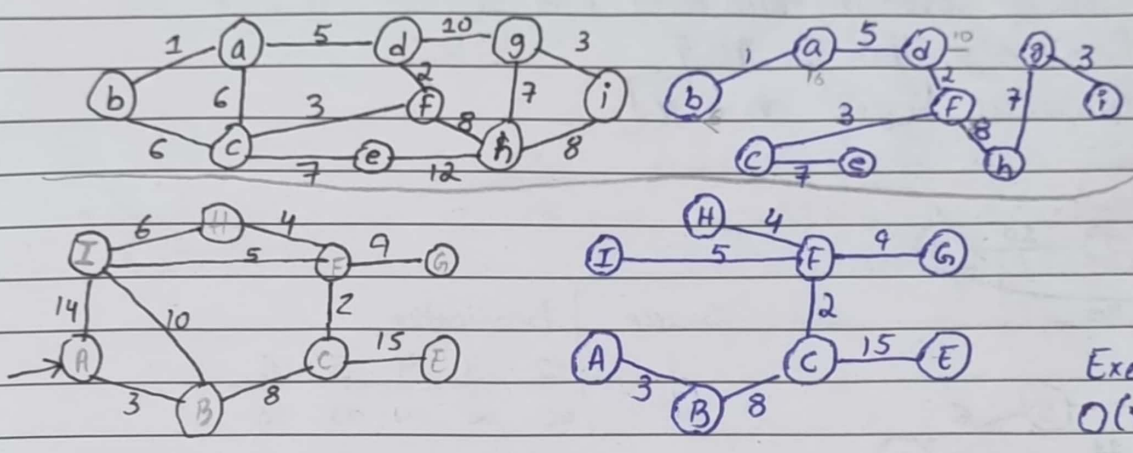


-) First we draw all vertecis then connect through edges with small weight & make sure no cycle created, in the end we get MST and edges will be (n-1)

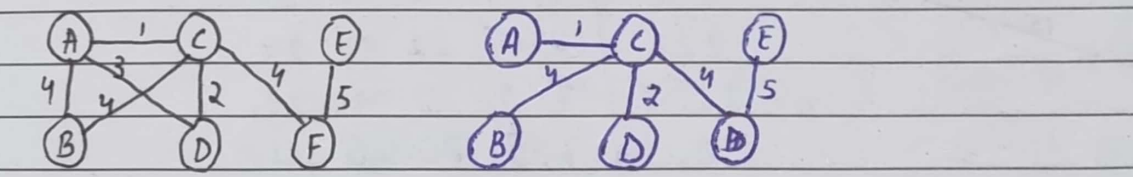
Date: _____

Prim's Algorithm: Source vertex is given. Source only check neighbour.
Graph shouldn't be disconnected. -) To find MCST -)Edges = V-1

Start with any vertex. Check its edges take less weighted edge move to next vertex, then check next edges and there weight also keep in mind of previous edges and there weight. No cycle created.



Execute time(E) = O(lg|V|)
O(V lg|V| + E lg V).



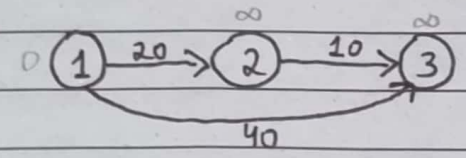
Growing the Tree: Building a tree by adding edges.
How to find next edge? Identify the next edge, which is smallest weighted edge
How to efficiently find the next edge?
Use efficient algo or data structure like priority queue.

Shortest Paths: Given a weighted Graph and two vertices u and v, we want to find a path of minimum total weight between u and v.
-)Length of path is the sum of weight of its edges.

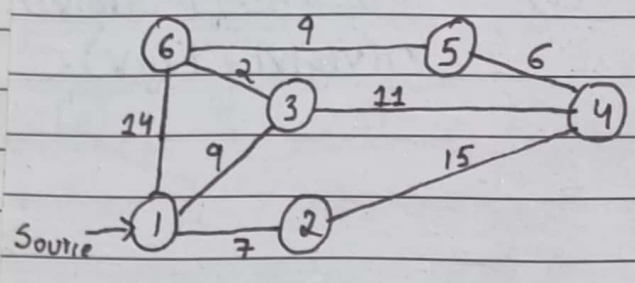
Application: Internet package routing, Flight reservations, Driving direction.

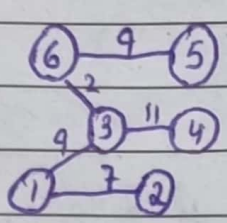
Date: _____

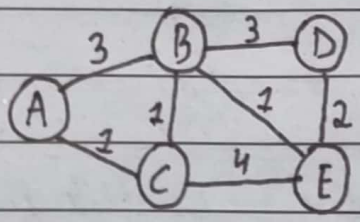
Dijkstra's Algorithm: No negative edge weights. (Single Source Shortest) Path
Similar to breadth-first search.
-> Grow a tree gradually, advancing from vertices taken from a queue.
Also similar to Prim's algorithm for MST
-> Use a priority queue keyed on d[v].

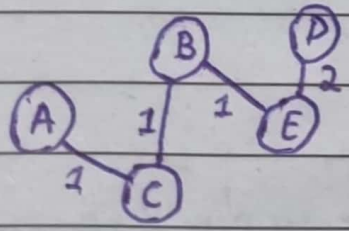


Source	Destination				
1	2	3	4	5	6
	∞	∞	∞	∞	∞
1-2	7	9	∞	∞	14
1,2,3	7	9	22	∞	14
1,2,3,6	7	9	20	∞	11
1,2,3,6,5	7	9	20	20	11
1,2,3,6,5,4	7	9	20	20	11





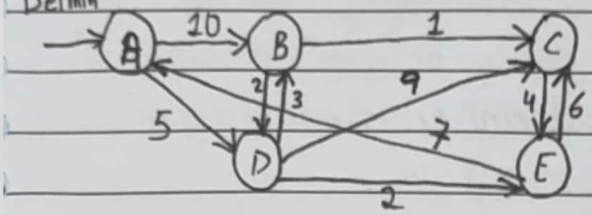




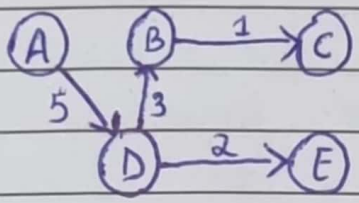
A	B	C	D	E
0	∞	∞	∞	∞
A,C	3	1	∞	∞
A,C,B	2	1	∞	5
A,C,B,E	2	1	5	3
A,C,B,E,D	2	1	5	3

Date: _____

Delmin

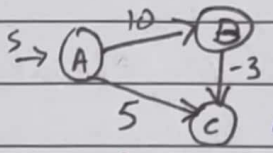


A	B	C	D	E
0	∞	∞	∞	∞
A,D	10	∞	(5)	∞
A,D,E	8	14	(5)	(7)
A,D,E,B	(8)	13	(5)	(7)
A,D,E,B,C	(8)	(9)	(5)	(7)



-ve weigh Edge

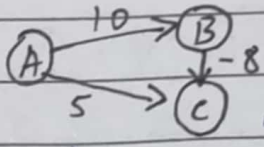
Working



A-B-C=7

A	B	C
0	∞	∞
AC	10	(5)
ACB	(10)	(5)

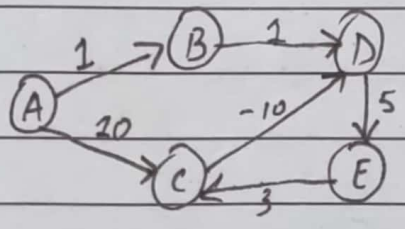
Not Working



A-B-

A	B	C
0	∞	∞
AC	10	(5)
ACB	(10)	(5)

(Q) What is the shortest path a to e?



A	B	C	D	E
0	∞	∞	∞	∞
AB	(1)	10	∞	∞
ABD	(1)	10	(2)	∞
ABDE	(1)	10	(2)	(7)
ABDEC	(1)	(10)	(2)	(7)

According to Dijkstra it 7, but it can be 5 by using path A-C-D-E.

Date: _____

Dynamic Programming

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to sub-problems. AND. The subproblems are overlapping.

It divides the problem into series of overlapping subproblems.

Possiblity is subproblem may repeat overlapping problem.

If we have overlapping in divide & Conquer we solve it again & again but in dynamic programming we solve it once & save it. This is called memorization

→ Solve it in bottom up fashion

→ Storing solution in array & mapping it if when we get similar subproble

Three main characteristics: ~~→Dp~~

1) Optimal substructure. 2) Overlapping subproblem 3) Memorization.

Steps in DP: Characterize structure of an optimal solution.

Define value of optimal solution recursively.

Compute optimal solution values either top-down with caching or bottomup in to

Construct an optimal solution from computed values.

More efficient than 'brute-force methods' which solves the same subprob over & over again.

↳All pair shortest path. Date: _____

Floyd Warshall's Algorithm:

Directed Graph: A graph whose every edge is directed is called directed graph or digraph.

Transitive Closure: The TC provides reach ability information about a digraph.

↳ We can perform DFS/BFS starting at each vertex.

- Perform traversal starting at the ith vertex
- Give info about the vertices reachable from the i^{th} vertex.

Drawback: This method traverses the same graph several times

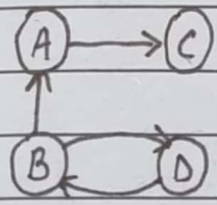
Efficiency: $(O(n(n+m)))$

Alternatively, we can use dynamic programming: the Warshall's Algorithm.

If we use dijkstra so Time Complex $V E \log V$ Worst case $V^3 \log V$

" " " Bellmanfor " " " VE " " V^4

Warshall's Algo: to find transitive closure. Not a direct path we can go from one vertex to another using path which is not directly connected.



$R(0)$ = Basic Matrix (1 where vertex is directly connected)

	A	B	C	D
A	0	0	1	0
B	1	0	0	1
C	0	0	0	0
D	0	1	0	0

$R(1)$ — 1st row & column remain same. A is intermediate.

	A	B	C	D
A	0	0	1	0
B	1	0	1	1
C	0	0	0	0
D	0	1	0	0

$R(2)$ — A & B are intermidiate. 2nd row/column remain sem.

	A	B	C	D
A	0	0	1	0
B	1	0	1	1
C	0	0	0	0
D	1	1	1	1

$R(3)$ — A, B & C intermidiate. 3rd row/column remain same.

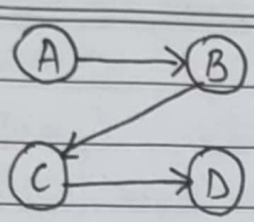
	A	B	C	D
A	0	0	1	0
B	1	0	1	1
C	0	0	0	0
D	1	1	1	1

$R(4)$ — ABCD are intermediate. 4th Row/Column remain same.

	A	B	C	D
A	0	0	1	0
B	1	1	1	1
C	0	0	0	0
D	1	1	1	1

Cont Lec 12....

Date: _____



R(0)

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

(R1)

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

(R2)

	A	B	C	D
A	0	1	(1)	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

R(3)

	A	B	C	D
A	0	1	(1)	(1)
B	0	0	1	(1)
C	0	0	0	1
D	0	0	0	0

R(4)

	A	B	C	D
A	0	1	1	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

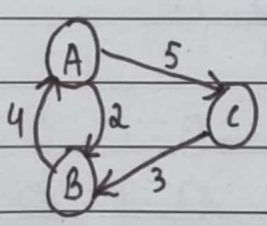
Floyd's Shortest Path:

Consider all paths start from one vertex & reaches the endpoint and does it again repeatedly all the vertices have been chosen as the starting & traversed than compare all the shortest paths and choose the one with least cost

) Dijkstra choose one path only. for reading the end goal.

Floyd's Warshall Algorithm: All pair shortest path.

∞ → when we can't reach directly, Alway zero in diagonal unless self loop



D(0)

	A	B	C
A	0	2	5
B	4	0	∞
C	∞	3	0

D(1)

	A	B	C
A	0	2	5
B	4	0	(9)
C	∞	3	0

5+4

@ 1st R & C will remain

D(2)

	A	B	C
A	0	2	5
B	4	0	9
C	(7)	3	0

AB

2nd R & C will remain same

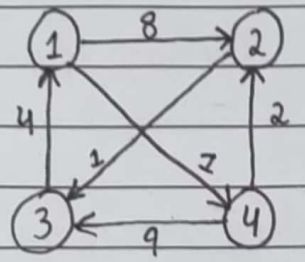
D(3)

	A	B	C
A	0	2	5
B	4	0	9
C	7	3	0

ABC

$$A^{k}[i,J] = \min\left\{ A^{k-1}[i,J],\ A^{k-1}[i,k] + A^{k-1}[k,J] \right\}$$

Floyd Warshall Algo T.C: $n(n^2) \to O(n^3)$ S.C: (n^2) Date: _____



$D(0)$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$D(1)$ — 1

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	(12)	0	(5)
4	∞	2	9	0

$D(2)$ — 1,2

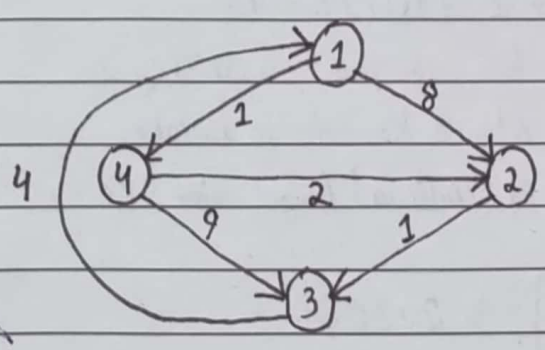
	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	(3)	0

$D(3)$ — 1,2,3

	1	2	3	4
1	0	8	9	1
2	(5)	0	1	(6)
3	4	12	0	5
4	(7)	2	3	0

$D(4)$ — 1234

	1	2	3	4
1	0	(3)	(4)	1
2	5	0	1	6
3	4	(7)	0	5
4	7	2	3	0



$D(0)$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$D(1)$ — 1

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	(12)	0	(5)
4	∞	2	9	0

$D(2)$ — 1,2

	1	2	3	4
1	0	8	(9)	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	(3)	0

$D(3)$ — 1,2,3

	1	2	3	4
1	0	8	9	1
2	(5)	0	1	(6)
3	4	12	0	5
4	(7)	2	3	0

$D(4)$ — 1,2,3,4

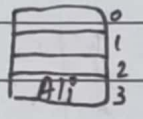
	1	2	3	4
1	0	(3)	(4)	1
2	5	0	1	6
3	4	(7)	0	5
4	7	2	3	0

Date: _____

Hash Tables:

Hashing: One way, can't revert means if we encrypt s^{th} we can't decrypt it
We use so that we can reduce the time complexity of searching.
Key Value → part of data. ∴ Authentication, Security.
Purpose of hashtable is that we get O(1) contant TC for searching, removing & inserting elements.

Ali → HashFunc → 3



% Comparision function
keys int > n.

Aik key multiple index par jaski → collision
Truncation: Take one part of data which is unique CS221011
Folding: Take whole part, make pattern CS221011 → 22+10+11 = 43
Anagram: Ali, Lia. Ali aur Lia ka same key ayegi tou uski position shi leyla.

Ali	Lia
359	593

k → contant → extra # number ka attach krai do jo number hai uskae upar shi polynomial laga ~~skt~~ skein.

Chaining:

k	h(k)		
221006	100%.10 = 0	0	221006 → 221009
221111	= 1	1	221111
221009	= 0	2	
221050	= 5	3	
221035	= 3	4	

Deletion mai: 0 par jei ga aur phr dekhai ga agr match kr sk rahi hai kae nhi.
→ Linear Probing: agar collision arha ho to jou shi index arhi + 1 i krtai
(0+2)%.10 = 2 (0+1)%.10 → data agae ha
→ Quadratic Probing: 1^2 2^2 3^2 → $(0+1^2)\%.10 = 1$, $(0+2^2)\%.10 = 4$.

Reshasing: Ek collision arha phr dukhai sae koi aur hashing technique arhi.
221006 par ek hi hash fn ja C221009
pr collision ayega tou dusra hash use.

Date:

Hashing: (Storing and Retriering Data in O(1) time)

* Search key (24, 52, 91, 67, 48, 83)
* Hash Table
* Hash Function (k mod 10, k mod n, Mid Square, Folding Method).

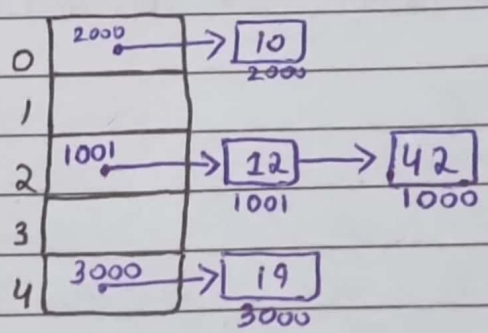
Collision Resolution Techniques

- Chaining (Open Hashing)
- Open Addressing (Closed Hashing)
  - -> Linear Probing
  - -> Quadratic Probing
  - -> Double Hashing.

Chaining:

keys: 42, 19, 10, 12

k mod 5



Advantage:

Deletion is easy just like linked list.

O(1) Best Case

O(1) Insertion

Disadvantage:

O(n) TC for searching.

↳ Worst case

Also space available in table

still it uses extra space.

i = Prob No / Collision No Date: _____

Linear Probing:

$R(k) = k \bmod 10$

$R'(k,i) = (R(k)+i) \bmod 10$ $h(k) = 23 \bmod 10 = 3$

Key: 43, 135, 72, 23, 99, 19, 82 $3+1 \bmod 10 = 4$

if collision occur then we will move to next available space

Advantage: No Extra Space.

Disadvantage: Search Time O(n), Deletion difficult.

Primary Clustering, Secondary Clustering

↳ When 2 or more elements are competing for same Prob sequence

2	19	0
		1
	72	2
	43	3
2	23	4
1	135	5
5	82	6
		7
		8
	99	9

Depend on 7 ~~& etc~~ Primary Cluster.

Q) The keys: 1, 3, 12, 4, 25, 6, 18, 20, 8 are inserted into empty hash table of length 10. When open addressing with hash function $H(i) = i^2 \bmod 10$ and linear probing. What is the resultant hash table and find the maximum prob value.

$H(i) = i^2 \bmod 10$

$1^2 \bmod 10 = 1$

$3^2 \bmod 10 = 9$

$12^2 \bmod 10 = 4$

$4^2 \bmod 10 = 6$

$25^2 \bmod 10 = 5$

$6^2 \bmod 10 = 6$ so we will insert in next available space i.e 7

$18^2 \bmod 10 = 4$, already element is there so place at 8

$20^2 \bmod 10 = 0$

$8^2 \bmod 10 = 4$, so next available space i.e 2

Prob Value		
1	20	0
1	1	1
9	8	2
		3
1	12	4
1	25	5
1	4	6
2	6	7
5	18	8
1	3	9

Date: _____

Quadratic Probing:

$h(k) = k \bmod 10$

$h'(k,i) = (h(k) + i^2) \bmod 10$

Keys: 42, 16, 91, 33, 18, 27, 36, 62 → 2,

↳ 6, already occupied so collision

$(h(k) + i^2) = 6 + 1^2 = 7$

$6 + 2^2 \bmod 10 = 0$

36	0
91	1
42	2
33	3
	4
	5
16	6
27	7
18	8
	9

Advantage: No Extra Space

↳ Primary Clustering Resolved

Disadvantage: Search O(n)

↳ Secondary clustering. No Guarentee of finding slot.

Double Hashing:

$h_1(k) = k \bmod 11$

$h_2(k) = 8 - (k \bmod 8)$

$(h_1(k) + i h_2(k)) \bmod 11$

if collision occur use $h_2(k)$

then $(h_1(k) + i h_2(k)) \bmod 11$

$1 + 1(3) \bmod 11 = \boxed{4} \rightarrow 45$

Advantage:

No Extra Space

No Primary & Secondary Clustering.

Disadvantage: O(n).

keys: 20, 34, 45, 70, 56

↳ 1, but occupied

$h(k) + i$

$h(k) + i^2$

$h_2(k) = 8 - (45 \bmod 8)$

$= 3$

$h_1(k) = 70 \bmod 11 = 4$

$h_2(k) = 8 - (70 \bmod 8) = 2$

$4 + 1(2) \bmod 11 = \boxed{6} \rightarrow 70$

$56 \bmod 11 = 1$

$8 - (56 \bmod 8) = 8$

$1 + 1(8) \bmod 11 = 9$, also occupied

$1 + 2(8) \bmod 11 = 6$ " "

$1 + 3(8) \bmod 11 = 3$

	0
34	1
	2
56	3
45	4
	5
70	6
	7
	8
20	9
	10

The shallower BST the better
Avg Case: O(logN) Best Case: O(N)

Date: _____

AVL Tree: Balanced BST

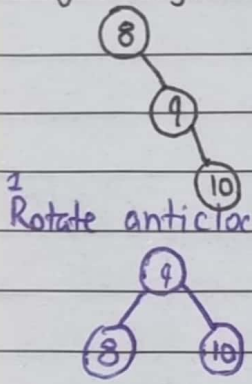
A BST that uses modified add and remove operations to stay balanced as element changes

When nodes are added/removed from the tree, if tree become unbalanced, repair the tree until balance is restored.

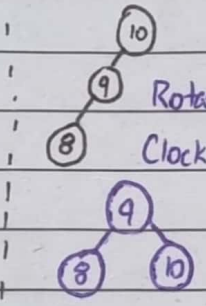
Rebalancing operations are relatively efficient (O(1))

Overall tree maintains balanced O(logN) height, fast to add/search.

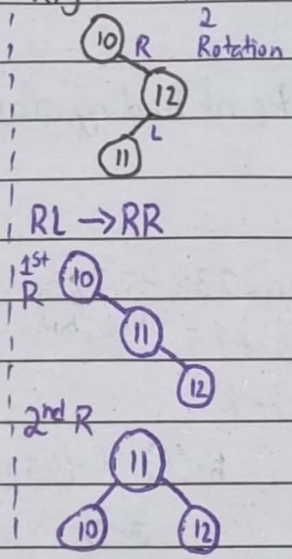
Right-Right:



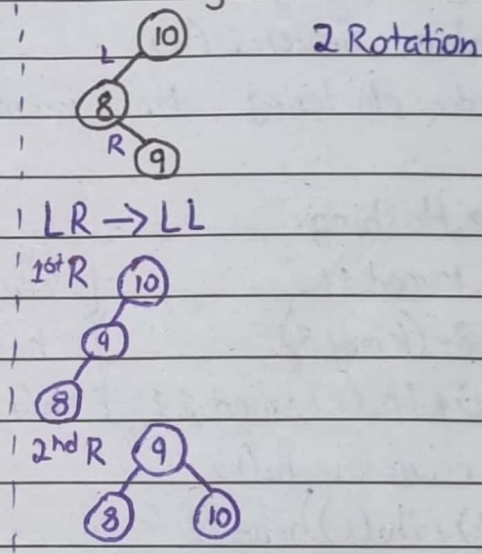
Left-Left



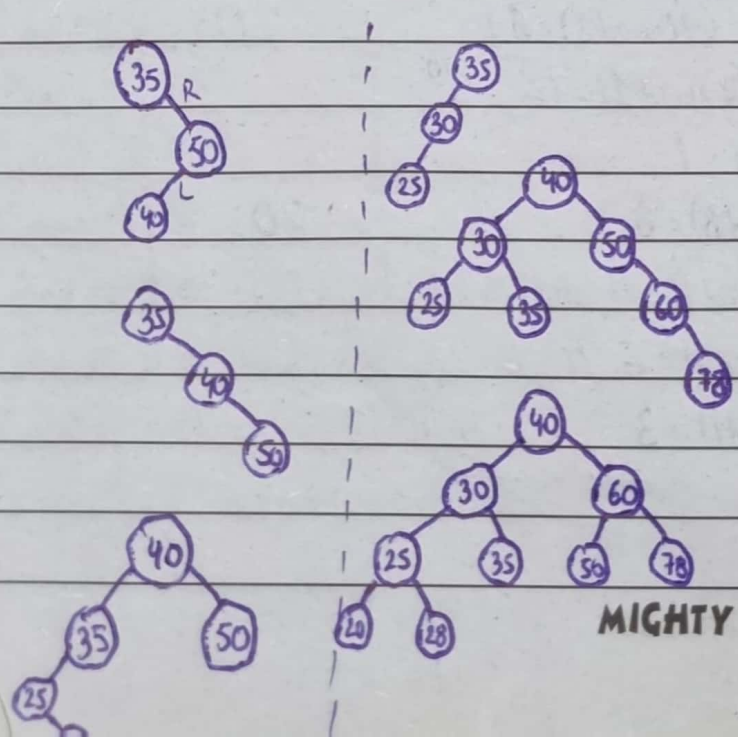
Right-Left



Left-Right



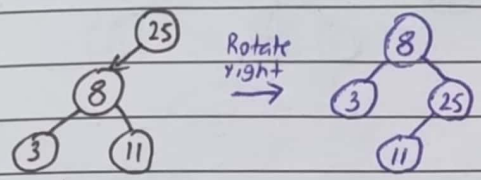
① 35, 50, 40, 25, 30, 60, 78, 20, 28.

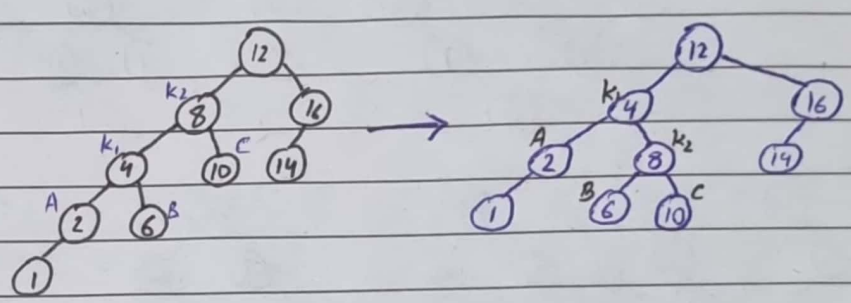


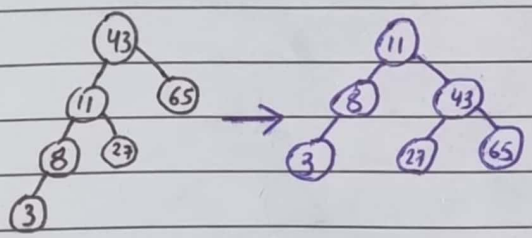
Date: _____

Right-Rotation (clockwise)

If a node has become out of balanced in a given direction, rotate it in opp direction.
rotation: A swap between parent and left or right child, maintaining proper BST ordering.

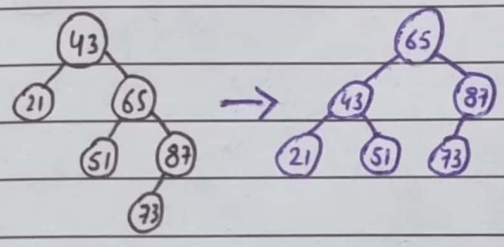




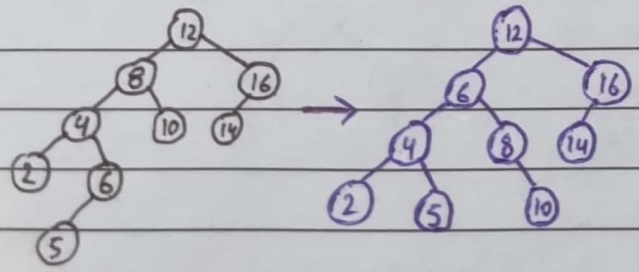


Left-Rotation (Anti-clockwise)

Right child k_2 becomes parent - Original parent k_1 demoted
k_2's original left subtree B (if any) is attached to k_1 as left subtree.



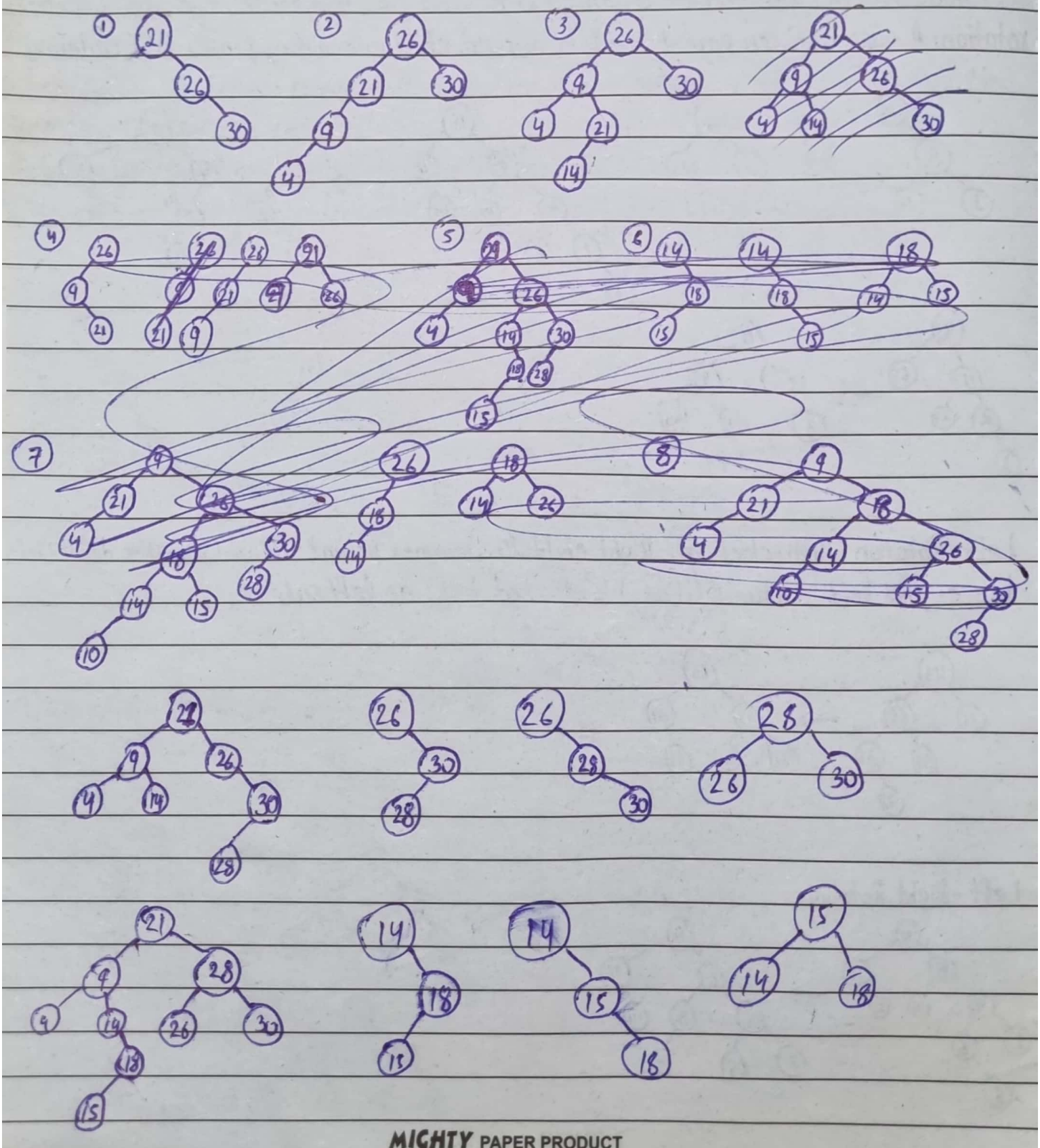
Left-Right Rotation



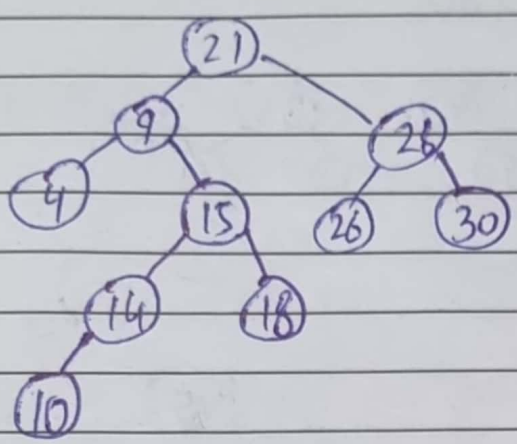
LST → LC → LST → RC
RST → LC → LST → RC
LST → RC → RST → LC
RST → RC → RST → LC

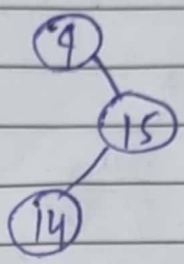
Date: _____

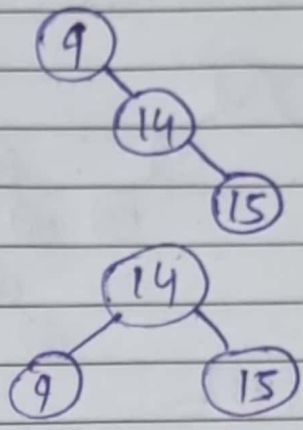
Q) 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

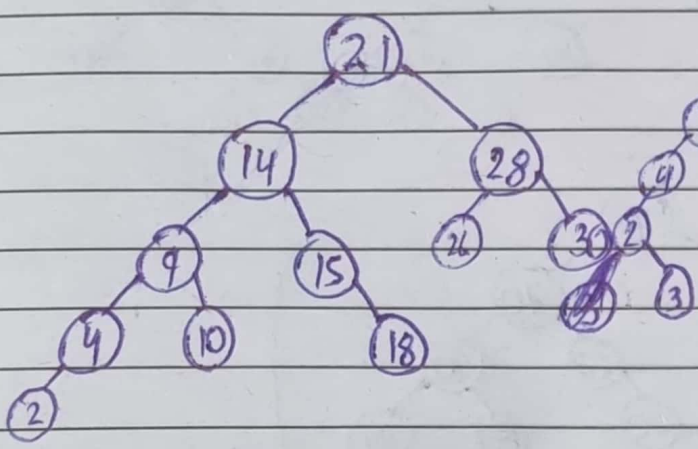


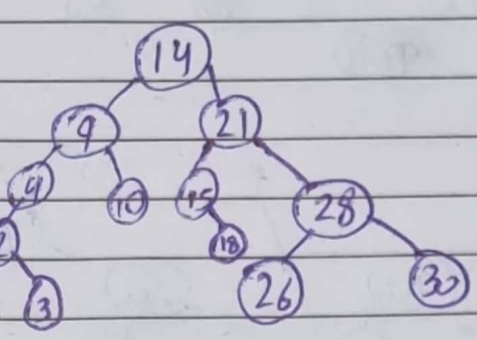
Date: _____

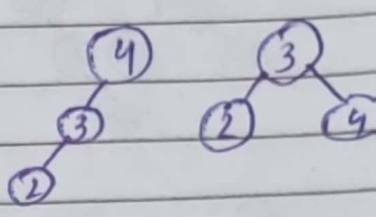


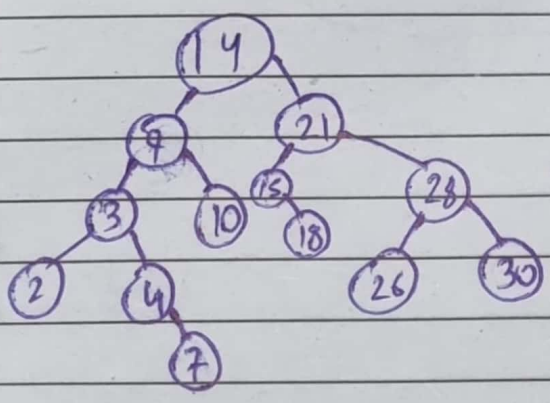


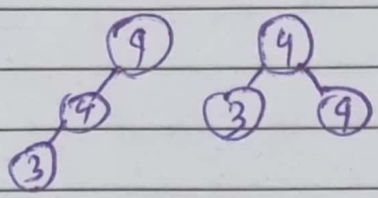


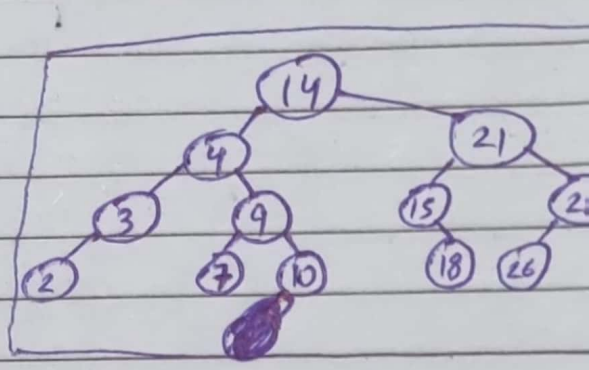






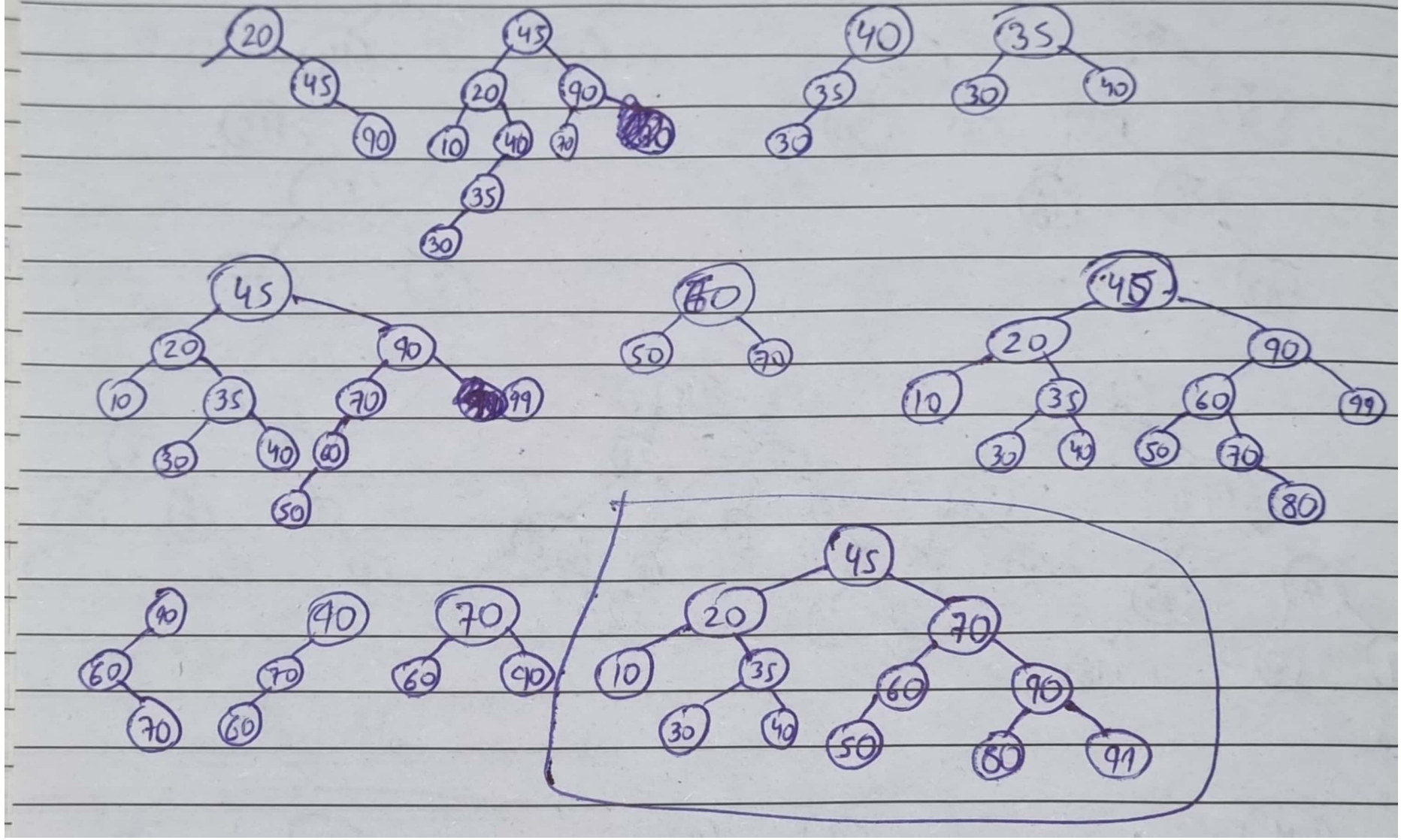






Date: _____

Q.) 20, 45, 90, 70, 10, 40, 35, 30, 99, 60, 50, 80



DAA Assignment #03

Date: _____

CS221004
Muhammad Anas

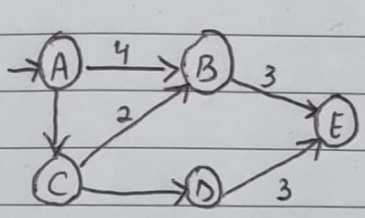
Q.1)

	W	V
[0]	= 0	
[1]	= 6	12
[2]	2	5
[3]	7	14
[4]	5	15
[1,2]	8	17
× [1,3]	13	not feasible
[1,4]	11	27
[2,3]	9	19

	W	V
[2,4]	= 7	20
[3,4]	= 12	29
[1,2,3]	= 15	not feasible
[1,2,4]	= 13	not feasible
[1,3,4]	= 18	" "
[2,3,4]	= 14	" "
{1,2,3,4}	= 20	not feasible

W = 12, V = 29
Optimal Solution

Q.2)

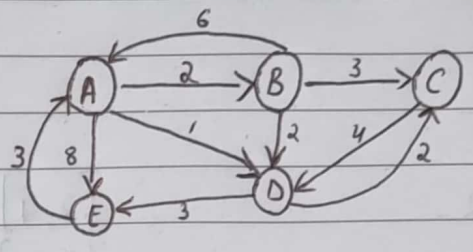


A	B	C	D	E
0	∞	∞	∞	∞
A-C	4	(1)	∞	∞
A-C-B	(3)	(1)	3	∞
A-C-B-D	(3)	(1)	(3)	7
A-C-B-D-E	(3)	(1)	(3)	(6)

Assignment #04 Date:

Q.1)

0	2	∞	1	8
6	0	3	2	∞
∞	∞	0	4	∞
∞	∞	2	0	3
3	∞	∞	∞	∞



Muhammad Ares
CS221004

(A)

R(1) A B C D E
A 0 2 ∞ 1 ∞
B 6 0 3 2 14
C ∞ ∞ ⊗ 4 ∞
D ∞ ∞ 2 0 3
E 3 5 ∞ ∞ 0

A(-)B

R(2) A B C D E
A 0 2 5 1 8
B 6 0 3 2 14
C ∞ ∞ 0 4 ∞
D ∞ ∞ 2 0 3
E 3 5 8 4 0

A(-)B(-)C

R(3) A B C D E
A 0 2 5 1 8
B 6 0 3 2 14
C ∞ ∞ 0 4 ∞
D ∞ ∞ 2 0 3
E 3 5 8 4 0

A(-)B(-)C(-)D

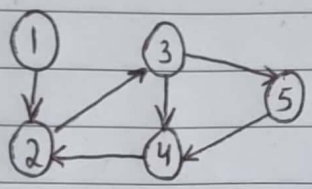
R(4) A B C D E
A 0 2 3 1 4
B 6 0 3 2 5
C ∞ ∞ 0 4 7
D ∞ ∞ 2 0 3
E 3 5 6 4 0

R(5) A B C D E
A 0 2 3 1 4
B 6 0 3 2 5
C 10 12 0 4 7
D 6 8 2 0 3
E 3 5 6 4 0

The shortest path.

Date: _____ 1

(Q.2)



R(0)

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	0	1	0	0	0
5	0	0	0	1	0

R(1)

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	0	1	0	0	0
5	0	0	0	1	0

1(-)2

R(2)

	1	2	3	4	5
1	0	1	(1)	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	0	1	(1)	0	0
5	0	0	0	1	0

1(-)2(-)3

R(3)

	1	2	3	4	5
1	0	1	1	(1)	(1)
2	0	0	1	(1)	(1)
3	0	0	0	1	1
4	0	1	1	(1)	(1)
5	0	0	0	1	0

1(-)2(-)3(-)4(-)5

R(4)

	1	2	3	4	5
1	0	1	1	1	1
2	0	1	1	1	1
3	0	1	1	1	1
4	0	1	1	1	1
5	0	1	1	1	1

DAA

Assignment #01 CS221004 Date:

Binary Search:

Space Complexity: $2n+2+2+2+2+2+2+2$

$2n+12$ $O(n)$

Time Complexity:

int l	1	1
int r	1	1
while	1	logn+2
int mid	1+1+1	3(logn+1)
if		
return	1	logn+1
if		
l=mid+1	2	2(logn+1)
r=mid-1	2	2(logn+1)
return	1	logn+1

$l <= r$

$0 \leq 8$

$5 \leq 8$

$7 \leq 8$

$8 \leq 8$

$9 \leq 8$

logn+2

$10\log n + 13$ $O(\log n)$

Selection Soxt:

Space Complexity: $2n+2+2+2+2+2$

$2n+10$

Date:

Selection Sort:

Space Complexity: $2n+2+2+2+2+2$

$\boxed{2n+10}$ $O(n)$

Time Complexity

int n	1	1
int i	1	1
i<n-1	1	n
i++	1	n-1
int minIndex	1	n-1
int j=i+1	1	(n-1)·1
j<n	1	n(n-1)
j++	1	(n-1)(n-1)
if		
minIndex=j	1	(n-1)(n-1)
int tem	1	1
arr[...]=arr[i]	1	1
arr[i]=temp	1	1

j=i+1

j<n i<n-1 n=3

1<3 0<2

2<3 1<2

3<3 2<2

$n^2-n-n+1$

n^2-2n+1

$1+1+n+n-1+n-1+n-1+n^2-n+n^2-2n+1+n^2-2n+1+1+1+1$

$\boxed{3n^2-n+4}$ $O(n^2)$

Date:

Matrix Multiplication (2D)

Space Complexity: $2n+2n+2+2+2+2n+2+2+2+2n$

$(8n+12)$ $O(n)$

OR $2n+2m+2+2+2+2p+2+2+2+2p$

$O(m*p)$

Time Complexity:

			$i<n$ $n=3$
int rA	1	1	$0<3$
int cA	1	1	$1<3$
int cB	1	1	$2<3$
int result	1	1	$3<3$
For int i	1	$n+1$	
$i<n$	1	n	
$i++$	1	n	
For int j	1	n	
$j<m$	1	$(m+1)\cdot n$	
$j++$	1	$n\cdot m$	
For int k	1	$n\cdot m$	
$k<p$	1	$n\cdot(p+1)\cdot m$	
$k++$	1	$n\cdot m\cdot p$	
result	3	$3(nmp)$	

$4+n+1+n+n+n+nm+n+nm+nm+nmp+nm+nmp+3nmp$

$(5nmp+4nm+5n+5)$

$O(n\cdot m\cdot p)$

M. Anas
CS221004 Assignment # 02 Date:

Merge sort is the most amenable to parallelization due to its natural divide-and-conquer structure, as it split the data into smaller parts and then merge them - This division & merging process can be easily paralized - Each sublist can be ~~stored~~ sorted independetly by different machines, then merging step combines the sorted sublists. The divide-conquer nature of merge sort naturally lends itself to parallel execution.

While on the other hand Quick sort is moderately ameanble, and Insertion & Heap sort is not very amenable to parallelization.

As Quick sort involves partitioning the data around a pivot element - Recursive nature makes it more complex. It can be parallelized ~~by~~ but handling recursion & load balancing requires careful design.

Insertion sort relies on comparing & shifting elements one by one Parallelizing process is complex as each step depends on previous one

Heap sort builds a heap (max heap or min heap) & repeatedly extract th root, so parallezing the heap construction is difficult - Heap property invorles swapping & adjusting the heap structure.

Therefore, if parallelization is a pirority, consider using merge sort for large data set.