UNIVERSITY PARTNER
UNIVERSITY OF WOLVERHAMPTON

iAcademy
Quality Education Improving Lives

HERALD COLLEGE
KATHMANDU

Academic Year	Module	Assessment Number	Assessment Type
S20	Introductory Data Structures and Algorithms (DipIT02)	A1	Not so easy question

[Not So Easy Question]

Student Id : [NP03A190299]

Student Name : [Yogesh Shrestha]

Section : [DC8]

Module Leader : [Mr. Prakash Gautam]

Submitted on : 06-04-2020

1-> Abstract data type are those type or class whose behaviour is defined by a set of value and a set of operations. It is usually implemented by an array.

The array and array type is also known as an abstract data type which hold a collection of elements that are accessible by an index. The elements which are stored in an array can be anything through primitive type for example: instances of classes which can be integer to more complex types.

Following are the areas in which data structure are applied extensively:

In Linear
-> Array
-> Link-list
-> Stacks
-> Queues

In Non-Linear
-> Trees
-> Graphs
-> Tables
-> Sets

2 → Asymptotic analysis of an algorithm refers to defining the mathmatical boundation / framing of its runtime performance. Therefore 1 sec is equal to 1×10^6 micro second.

	1 Sec	1 Min	1 Hour	1 Day	7 Day	15 Day	1 Month
$\log\log n^2$	$2^{5\times10^5}$	$2^{3\times10^7}$	$2^{1.8\times10^9}$	$2^{4.32\times10^{10}}$	[illegible]	$2^{6.048\times10^{11}}$	$2^{1.314\times10^{12}}$
$\sqrt{n}$	1×10^{12}	3.6×10^{15}	1.29×10^{21}	7.46×10^{21} [illegible]	3.6×10^{23}	1.6×10^{24}	6.7×10^{24}
$2n^3$	79	310	1216	3508	6712	8654	10903
2^n	19	25	31	36	39	40	41
$n!$	9	11	12	13	14	14	15

Hence, from the table the $\log n^2$ is distinguished to be the best time complexity algorithm in compared to others. When we analyze the algorith, ordered by slowest to fastest growing: ~~then or the by good time~~ then, the best or good time complexity is:

$\log\log n^2 > \sqrt{n} > 2n^3 > 2^n > n!$

3)> Solution

Pseudocode

```
largestInteger(n)
i = 2^(n-1) - 1;
Print i;
```

for 1 bit

$2^1 = 2 = 0, 1$

for 2 bit

$2^2 = 4 = 0, 1, 10, 11$

for 3 bit

$2^3 = 8 = 0, 1, 10, 11, 100, 101, 110, 111$

for n bit

2^n

for 32 bit

$2^{32} = 4294967296$

for the largest integer

$2^{n-1} - 1$

$= 2^{32-1} - 1$

$= 2147483647$

4 -> Solu

Given: $(x+3y)*/(3z-p+2^t)$

Converting infix to postfix using stack:

S.N	Scanned	Stack	Post fix	Description
1	(	(	(	Start
2.	x	(	x	
3	+	(+	x	
4	3y	(+	x3y	
5	)		x3y+	
6	/	/	x3y+	
7	(	/(	x3y+	
8	3z	/(	x3y+3z	
9	-	/(-	x3y+3z	
10	p	/(-	x3y+3zp	
11	+	/(-+	x3y+3zp	
12	2^t	/(-+^	x3y+3zp2^t	
13	)		x3y+3zp+^+/	End

∴ the postfix value is x3y+3z-p+^+/

5->. Stack - is used to perform Recursion because of the Last In First Out (LIFO) property. which ~~is used to store~~ works by storing return addresses of the function calls.

For example: Pile of group of weights used in Gym machines.

In 'C' code

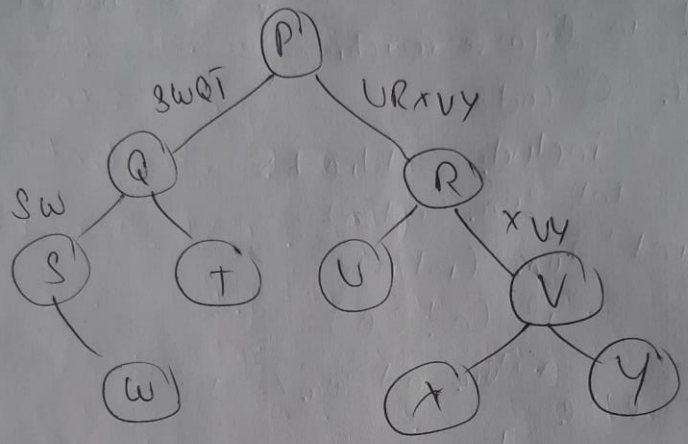
```
#include <stdio.h>
int sum(int n);
int sum (int n){
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}
int main(){
    int number, result;
    printf("Enter a positive integer");
    scanf("%d", &number);
    result = sum(number);
    printf("sum = %d", result);
    return 0;
}
```

6 → Solution.

Given:

Inorder traversal: SWQTPURXVY

Preorder Traversal: PQSWTRUVXY



7 → Solo

Given:

$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$

is $T(n) = n \log n$

Base step

If $n = 2$, then $T(n)$ $T(2) = 2^k$ $2 \log 2 = 2$

Thus, $T(2) = 2 \log 2$

Hypothesis step

Assuming $T(n) = n \log n$ is true if $n = 2^k$ for some integer $k > 1$.

Inductive step

If $n = 2^{k+1}$, then

$T(n) = 2T(2^{k+1}/2) + 2^{k+1}$

$= 2^1 (2^k) + 2^{k+1}$

$= 2(2^k \log 2^k) + 2^{k+1}$ [$\because T(n) = n \log n$]

$= 2^{k+1} ((\log 2^k) + 1)$

$= 2^{k+1} \log 2^{k+1}$

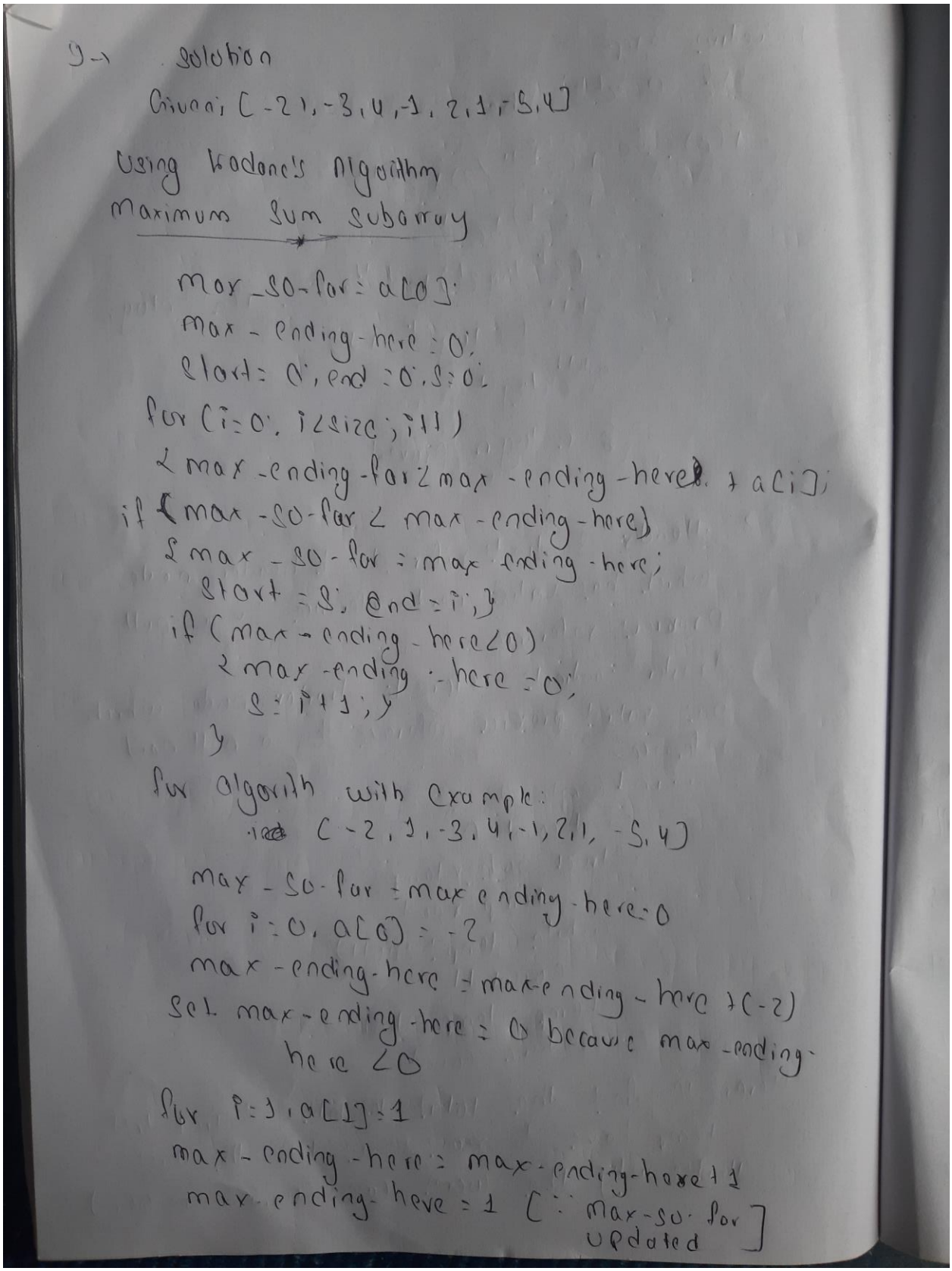
$= n \log n$ proved

2. -> Header sentinels refers to a header node which has valid next reference by a null previous reference.

Trailer sentinels refers to trailer node which has valid previous reference by a null next reference.

```
int length()
    int count = 0
    struct node *temp
        temp = root
    while (temp != null)
        count ++
        temp = temp->link
        return count
```

Here, it find the total length of link list temp have to move till the last that is why the worst-case scenario is O(n)



9→ Solution

Given: [-2, -3, 4, -1, 2, 1, -5, 4]

Using Kadane's Algorithm

Maximum Sum Subarray

```
max_so_far = a[0];
max_ending_here = 0;
start = 0, end = 0, s = 0;
for (i=0; i<size; i++)
{ max_ending_here = max_ending_here + a[i];
if (max_so_far < max_ending_here)
{ max_so_far = max_ending_here;
  start = s; end = i; }
if (max_ending_here < 0)
{ max_ending_here = 0;
  s = i+1; }
}
```

For algorith with Example:

Given: (-2, 1, -3, 4, -1, 2, 1, -5, 4)

max_so_far = max_ending_here = 0

for i = 0, a[0] = -2

max_ending_here = max_ending_here + (-2)

Set max_ending_here = 0 because max_ending_here < 0

for i = 1, a[1] = 1

max_ending_here = max_ending_here + 1

max_ending_here = 1 [∵ max_so_far updated]

for i = 2, a[2] = -3
max-ending-here = max-ending-here + (-3)
max-ending-here = 0

for i = 3, a[3] = 4
max-ending-here = max-ending-here + (4)
max-ending-here = 4
max-so-far = 4

for i = 4, a[4] = -1
max-ending-here = max-ending-here + (-1)
max-ending-here = 3

for i = 5, a[5] = 2
max-ending-here = max-ending-here + (2)
max-ending-here = 5

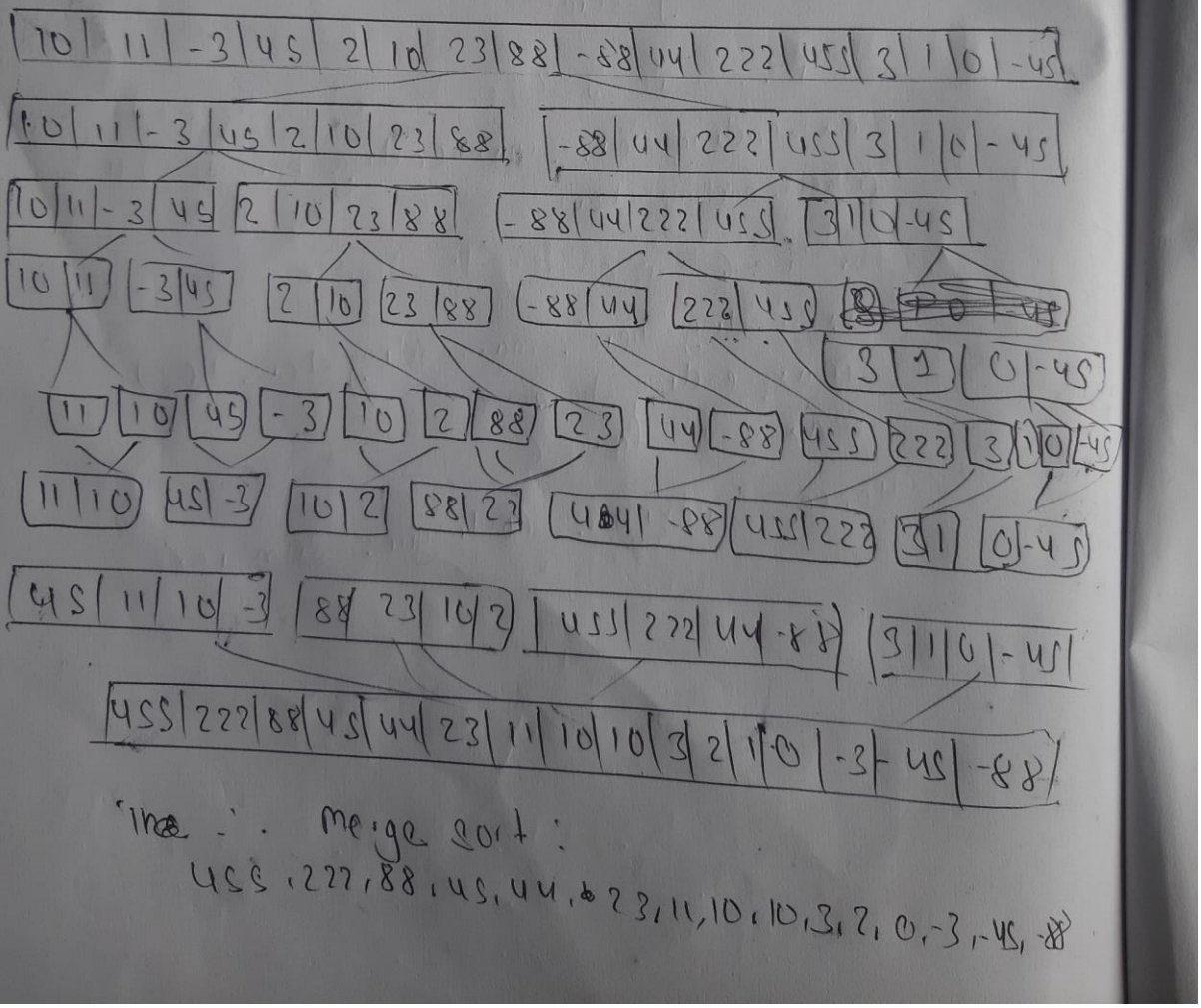
for i = 6, a[6] = 1
max-ending-here = max-ending-here + (1)
max-ending-here = 6
max-so-far = 6

Hence, maximum contiguous sum is 6.

10 → No, Quicksort is not a stable algorithm because its swipping is done accroding to ~~five~~ Pivot's position. It can be stable if two objects with equal keys are appear in the same order in sorted output as the appear in the input array to be sorted.

Merge Sort

~~10 11 3 45 2 10~~



Ans :. merge sort :

455, 222, 88, 45, 44, 23, 11, 10, 10, 3, 2, 1, 0, -3, -45, -88

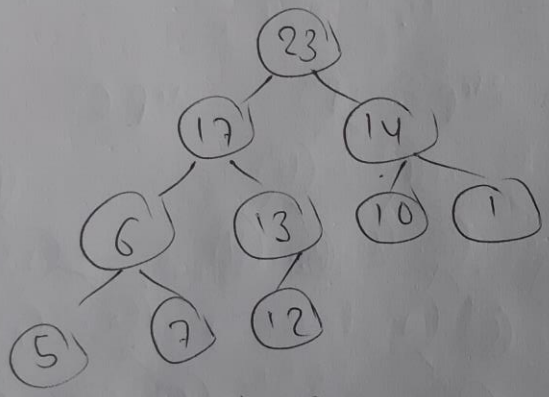
11→ The maximum and minimum numbers of elements in a heap of height are:

When the leaf node of is at height 1

maximum : $2^h - 1$

minimum : $2^h - 1$

→ Ans. the array with value <23, 17, 14, 6, 13, 10, -1, 5, 7, 12> is not a max-heap because the parent '7' is 6 in the array which violates the max-heap property.



∴ 6 < 7

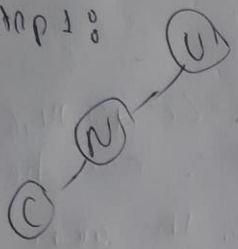
13→. Sol:

Given: UNCOPYRIGHTABLES

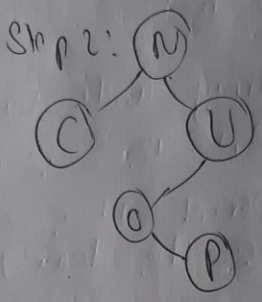
Now:

Constructing AVL tree

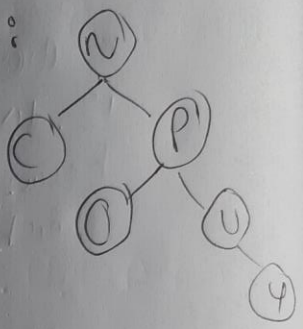
Step 1:



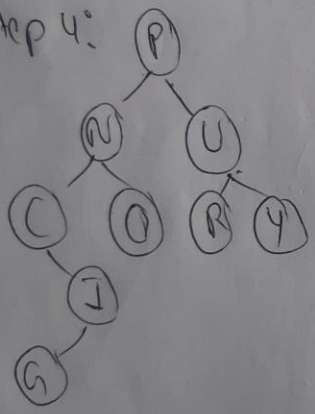
Step 2:



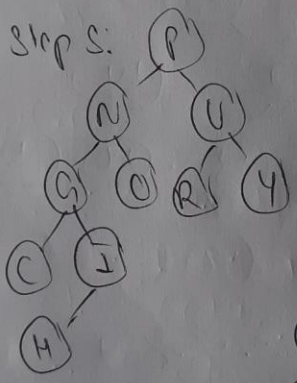
Step 3:



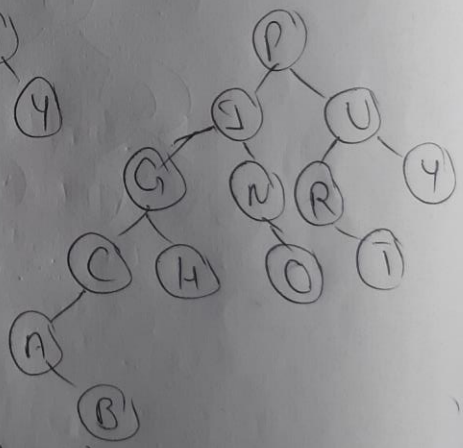
Step 4:



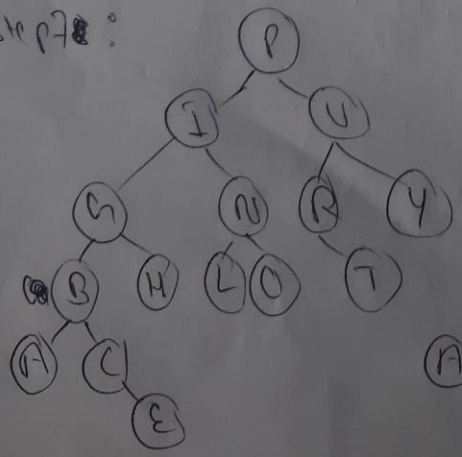
Step 5:



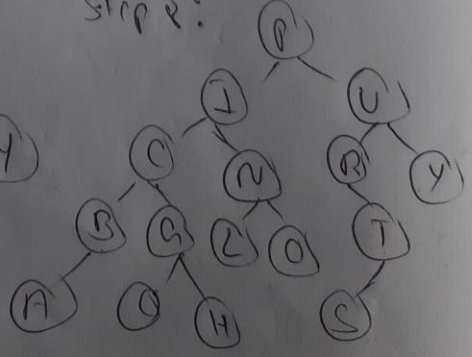
Step 6:



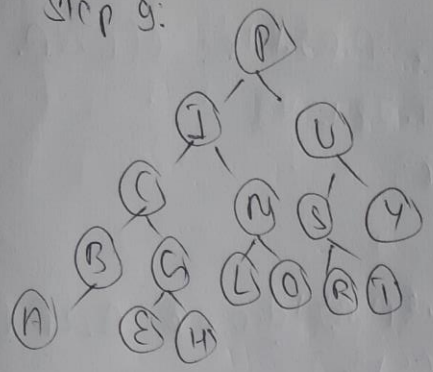
Step 7:



Step 8:



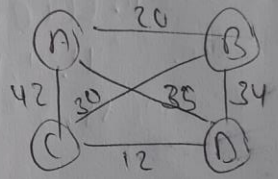
Step 9:



14 -> Solu

Given figure:

Temp



$S = \{A\}$ which is last city.

The 3 possible ways to get A are:

$C(\{B\}, \{A\}) = \min[C(A, \{\}) + d_{BA}] = 0 + 20 = 20$

$C(\{C\}, \{A\}) = \min[C(A, \{\}) + d_{CA}] = 0 + 42 = 42$

$C(\{D\}, \{A\}) = \min[C(A, \{\}) + d_{DA}] = 0 + 35 = 35$

$S = \{A, B\}, \{A, C\}, \{A, D\}$

$C(C, \{A, B\}) = \min[C(B, \{A\}) + d_{CB}] = 20 + 30 = 50$

$C(D, \{A, B\}) = \min[C(B, \{A\}) + d_{DB}] = 20 + 34 = 54$

$C(\{B\}, \{A, C\}) = \min[C(C, \{A\}) + d_{BC}] = 42 + 30 = 72$

$C(\{D\}, \{A, C\}) = \min[C(C, \{A\}) + d_{DC}] = 42 + 12 = 54$

$C(\{B\}, \{A, D\}) = \min[C(D, \{A\}) + d_{BD}] = 35 + 34 = 69$

$C(\{C\}, \{A, D\}) = \min[C(D, \{A\}) + d_{CD}] = 35 + 12 = 47$

$S = \{C, B, A\}, \{D, B, A\}, \{D, C, A\}$

$C(D,\{C,B,A\}) = \min[C(C,\{B,A\}) + d_{DC}, C(B,\{A,C\}) + d_{DB}]$

$= \min[50 + 12, 72 + 34]$

$= \min[62, 106]$

$= 62$

$C(C,\{D,B,A\}) = \min[C(D,\{B,A\}) + d_{CD}, C(B,\{A,D\}) + d_{CB}]$

$= \min[54 + 12, 69 + 30]$

$= \min[66, 99]$

$= 66$

$C(B,\{D,C,A\}) = \min[C(D,\{A,C\}) + d_{BD}, C(C,\{A,D\}) + d_{BC}]$

$= \min[54 + 34, 42 + 30]$

$= \min[88, 72]$

$= 72$

$S = \{D,C,B,A\}$

$C(A,S) = \min[C(D,\{C,B,A\}) + d_{AD}, C(C,\{D,B,A\}) + d_{AC}, C(B,\{D,C,A\}) + d_{AB}]$

$= [62 + 35, 66 + 42, 72 + 20]$

$= \min[97, 108, 92]$

$= 97$

∴ Required path ABCDA
ACDBA

```
Is ->. Electricity(data)
  if (Q. isEmpty)
    Q. Enqueue(data)
    Q. front = data;
  else
    if (Q.length != array.length-1)
    if(data == Q.front)
      dequeue(Q);
    if (Q.length == array.length-1)
      dequeue(Q);
      else Enqueue(data);
    else
      dequeue(data);
    return

Enqueue(Q, x)
  Q.[Q.back] = x
  if (Q.back == Q.length)
    Q.back = 0
    else Q.back = Q.back+1

  dequeue(Q)
    x = Q[Q.front]
    if (Q.front == Q.length)
      Q.front == 0
    else Q.front = Q.front+1
      return x.

  OverFlow()
    if array.length == queue.length
      return true
      else false
: Time complexity is O(1)
```

16 -> Solution.

Given hash function $h(n) = n$

the order of array is

0	1	2	3	4	5	6	7	8
9	18		12	13	14	4	21	

for linear provision

$h(n) = n \% 9$

for A - 9, 14, 4, 18, 12, 3, 21

key (n)	Position ($h(n) = n \% 9$)
9	$9 \% 9 = 0$
14	$14 \% 9 = 5$
4	$4 \% 9 = 4$
18	$18 \% 9 = 0$
12	$12 \% 9 = 3$
3	$3 \% 9 = 3$
21	$21 \% 9 = 3$

0	1	2	3	4	5	6	7	8
9	18		12	4	14	3	21	

Here, in 4 index we have 4, but in array we have 3. So, it is not suitable

B - 12, 3, 14, 18, 4, 9, 21

key (n)	Position ($h(x) = x \% 9$)
12	$12 \% 9 = 3$
3	$3 \% 9 = 3$
14	$14 \% 9 = 5$
18	$18 \% 9 = 0$
4	$4 \% 9 = 4$
9	$9 \% 9 = 0$
21	$21 \% 9 = 3$

0	1	2	3	4	5	6	7	8
18	9		12	3	14	4	21	

Here, In 0 index we have 18, but in array we have 9. so, it also can't be the possible one.

C - 12, 14, 3, 9, 4, 18, 21

Key(n)	Position (h(n) : n%9)
12	12%9=3
14	14%9=5
3	3%9=3
9	9%9=0
4	4%9=4
18	18%9=0
21	21%9=3

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

It is one of the possible answer

D - 9, 12, 14, 3, 4, 21, 18

Same as above 'C' in the index 6 there is 21 and the original array is also 21 it is possible answer

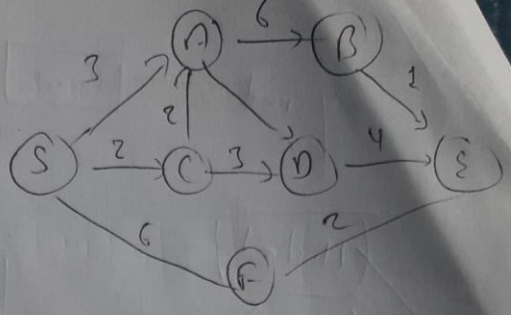
E - ~~Same as~~ 12, 9, 18, 3, 14, 21, 4

Same as above, in the index 6 there is 21. but in our ~~org~~ original array it is 4 so, it is not possible answer.

∴ the possible answer is C and D.

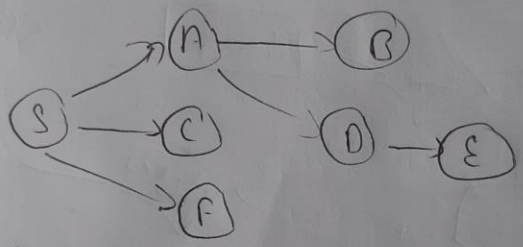
17 → Soln

Given,



In Dijkstra's algorithm, it can visit following vertices in order.
S, C, A, D, F, E }.

So, the parent of each node should be like this:



Dijkstra algorithm only removes from the priority queue |v| times. Removal takes of $O(\log |v|)$ time for a total of $O(|v| \log(v))$ time for all vertex removals.

For the shortest path to vertices.

Vertices	S	A	B	C	D	E	F
S	0_S	3_S	∞_S	2_S	∞_S	∞_S	6_S
A	0_S	3_S	∞_S	2_S	5_P	∞_S	6_S
B	0_S	3_S	9_A	2_S	4_A	∞_S	6_S
C	0_S	3_S	9_A	2_S	4_A	8_D	6_S
D	0_S	3_S	9_A	2_S	4_A	8_D	6_S
E	0_S	3_S	9_A	2_S	4_A	8_D	6_S
F	0_S	3_S	9_A	2_S	4_A	8_D	6_S

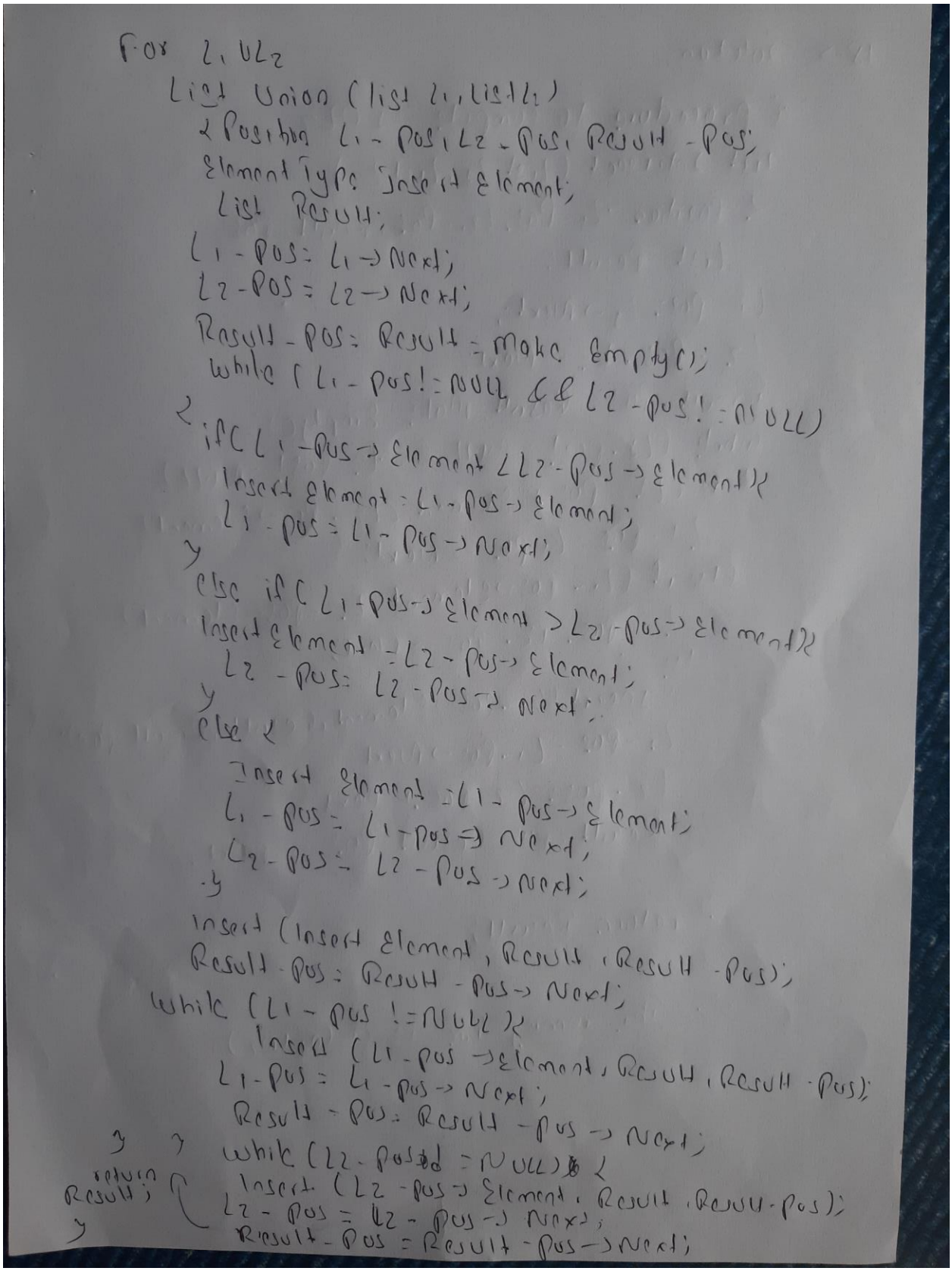
∴ Path : S, A, D, E

18-> Solution

a Procedure to compute $L_1 \cap L_2$

```
List Intersect (List L1, List L2)
{ Position L1_Pos, L2_Pos, Result_Pos;
  List Result;
L1_Pos = L1 -> Next;
L2_Pos = L2 -> Next;
Result_Pos = Result = MakeEmpty();
while (L1_Pos != NULL && L2_Pos != NULL)
{ if (L1_Pos -> Element < L2_Pos -> Element)
    L1_Pos = L1_Pos -> Next;
  else if (L1_Pos -> Element > L2_Pos -> Element)
    L2_Pos = L2_Pos -> Next;
  else
  { Insert(L1_Pos -> Element, Result, Result_Pos);
    L1_Pos = L1_Pos -> Next;
    L2_Pos = L2_Pos -> Next;
    Result_Pos = Result_Pos -> Next;
  }
}
return Result;
}
```

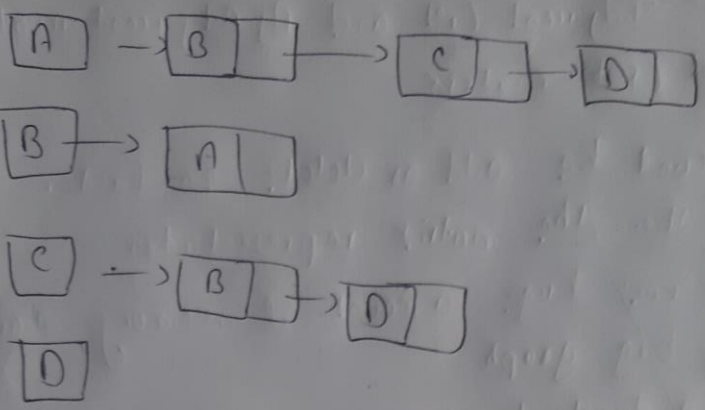
∴ the running time is $O(n_1 + n_2)$.



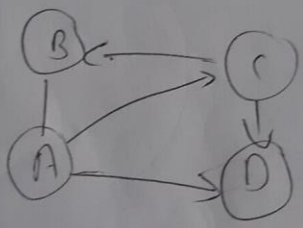
```
For L1 UL2
   List Union (List L1, List L2)
   {Position L1_pos, L2_pos, Result_pos;
    ElementType Insert Element;
    List Result;
    L1_pos = L1->Next;
    L2_pos = L2->Next;
    Result_pos = Result = MakeEmpty();
    while ( L1_pos != NULL && L2_pos != NULL)
    {
    if(L1_pos->Element < L2_pos->Element){
      Insert Element = L1_pos->Element;
      L1_pos = L1_pos->Next;
    }
    else if( L1_pos->Element > L2_pos->Element){
    Insert Element = L2_pos->Element;
      L2_pos = L2_pos->Next;
    }
    else {
       Insert Element = L1_pos->Element;
       L1_pos = L1_pos->Next;
       L2_pos = L2_pos->Next;
    }
    Insert (Insert Element, Result, Result_pos);
    Result_pos = Result_pos->Next;
    while (L1_pos != NULL){
        Insert (L1_pos->Element, Result, Result_pos);
        L1_pos = L1_pos->Next;
        Result_pos = Result_pos->Next;
    } } while (L2_pos != NULL){
        Insert (L2_pos->Element, Result, Result_pos);
        L2_pos = L2_pos->Next;
        Result_pos = Result_pos->Next;
    return Result;
    }
```

1) → Solution

Given:



a → Directed graph



b) Adjacency Matrix

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

Edges: {(A, B), (A, C), (A, D), (B, A), (C, B), (C, D)}

c) - Adjacency matrix

Pros - Adjacency matrix is fast to check whether edge between two nodes are exists or not.

- Add or delete an edge can be done in O(1) time.

Cons -

- Adjacency matrix consumes a lot of memory for storing big graph.
- The effort for adding or deleting nodes requires huge efforts
- It redundant the information for undirected graph

Adjacency List

Pros -

- Adjacency list allows efficients space to store graph.
- The new edges can be added in O(1) time.

Cons

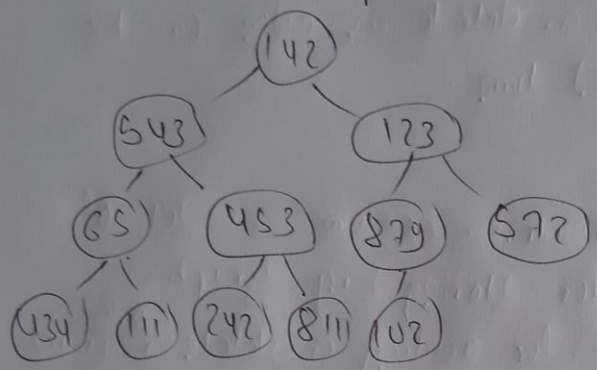
- It does not allow to make an efficient implementation if dynamically change of vertices number is required.
- Adding or deleting edge to it is not hard for adjacency matrix

20-> The running time of heapsort for presorted input is O(nlogn).

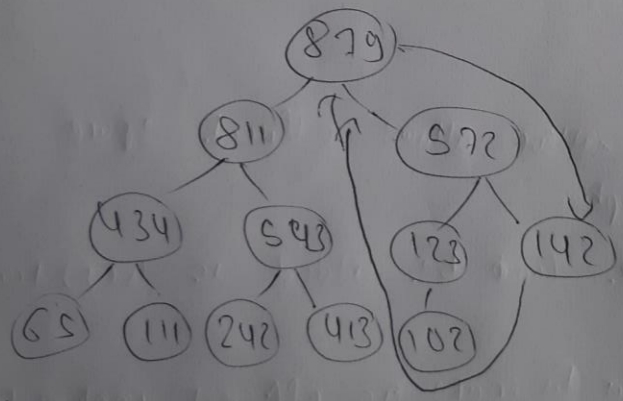
Given data:

142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102

Step 1: Build the heap



Now, turning into maximum heap.



Step 2: