Design And Analysis Of Algorithms.

Date: M T W T F S S

(All codes of these algorithms have been uploaded on gcr along with assignment)

K20-1052
BSE-5B
S.M. Hassan Ali

Q.1

O(n^on) solution.

```
int arr[] = {elements here};
int sum = 0, max = 0;

for(int i=0; i<arr.len; i++){
    if (arr[i] >=0){
        sum = sum + arr[i];
    }
    if( arr[i] <0){
        if(sum > max){
        max = sum;
        sum = 0;
        }
    }
}

if(sum > max){
    max = sum;
}
```

just a loop means linear time taken.

K20-1052.

Date:

M	T	W	T	F	S	S

O(n²) solution.

```
int arr[] = {some elements};
int sum, max;
int arr2[]; int j;
for(i=0; i<arr.len; i++){
  if( arr[i] >= 0){
    arr2[j++] = arr[i]
  }
if(arr[i]< 0){
for(k=0; k<j; k++){
  sum = sum + arr2[k];
}
if (sum > max){
  max = sum; sum=0;
  arr2[] = null; j=0;
}}
}
if(sum > max){
    max = sum;
```

nested loop leads to O(n²)

RC

Signature

k20-1052

Date:

Q.2. O(nlogn)

A[] = {3,1,2,4}
B[] = {5,1,6,2}

Use mergesort on Array B, which will cost O(nlogn). Strategy is to apply Binarysearch on Array B. using a for loop, which will find element y as y = x - A[i]. If y exists, means we have such that x = A[i] + A[j]

x = 10 y = 0 n = 4

```
for(i=0; i<n; i++){
        y = x - A[i];          // y = 10 - 4 = 6   A[3]
result = Binarysearch(B[], 0, n-1, y);
if(result == 1){
//Print index of Array A and B
}
}
```

for loop followed by binarysearch = n x logn
Merge sort = nlogn

nlogn + nlogn = O(nlogn)

K20-1052.

Date:

M	T	W	T	F	S	S

O(n)
Strategy is to store elements of array B in Hash Table that will take mean while insertion/searching in Hash Table takes O(1).

A[] = {3, 1, 2, 4}
B[] = {5, 1, 6, 2}

O(n) for a single loop.

```
// Array B[] already stored in Hash Table. size O(n)
Bool FindSum(int A[], int n, int x){
    int y;
    for(i=0; i<n; i++){
        y = x - A[i];
        if(HT.search(y) == true){ return 1 }
    }
    return -1
};
```

→ Here it will compare the value y with the elements of B stored in HT. It will take O(1) and loop will take

K20-1052.

Date:

M	T	W	T	F	S	S

Q.3 O(log n)

→ Strategy: use of binary search and a additional condition for finding mid.

```
int arr[] = {elements}
int s = binarysearch(arr, 0, arr.length-1, x)
```
element for finding

```
int binarysearch(int arr, int l, int r, int x){
  if ( r >= l){        // for checking element within range.
  int mid = l + (r-l)/2;

if ( arr[mid] == x && mid == x){
    // print mid = x
    return mid };

if ( arr[mid] > x){
    return binarysearch(arr, l, mid-1, x); }
  else {
  return binarysearch(arr, mid+1, r, x);
  }

  return -1;
}
```


RC

Signature

K20-1052

Date:

M	T	W	T	F	S	S

Q.4 O(n) time and space O(1)
strategy: Use of two pointers, if arr[i] is not zero swap element present at j++ else only i move forward.

```
int arr[] = {some elements} // {1,0,4,2,0}
int j=0;
for(int i=0; i<arr.len; i++){
    if(arr[i] != 0){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        j++;
    }
}
// just print the elements
```

Time complexity O(n) as linear traversing.
and space complexity O(1) as constant space is being used that is not changed

K20-1052

Date:

M	T	W	T	F	S	S

Q.5a Binary and Jump Search.

Binary Search arr[] = {0, 1, 3, 8, 10, 16, 20, 28}

n = 8 l = 0 r = 7 x = 10

1- mid = l + (r-l)/2 = 0 + (7-1)/2 = 3

mid

arr[3] != 10

arr[mid] < x

so → bs(arr, mid+1, r, x)

2- l = mid+1 = 3+1 = 4, r = 7, x = 10

mid = 4 + (7-4)/2 = 5

arr[5] != 10

arr[mid] > 10

so → bs(arr, l, mid-1, x);

3- l = 4, r = mid-1 = 5-1 = 4, x = 10

mid = 4 + (4-4)/2 = 4

arr[4] == 10

10 == 10

found.

Page No.

RC

Signature

K20-1052.

Date:

M	T	W	T	F	S	S

Jump Search arr[] = {0, 1, 3, 8, 10, 16, 20, 28}
n = 8 x = 10

1- Step = [sqrt(7)] → 2

prev = 0

while (arr[min(2, 7) - 1] < 10)
(step, n, x)
prev = step → 2
step = step + sqrt(n) → 4

// finding the max range in which the element can be

2-
while (arr[min(4, 7) - 1] < 10)
prev = step → 4
step = step + sqrt(n) → 6

3-
while (arr[min(6, 7) - 1] < 10)

while (arr[4] < 10)
(prev)

// for linear search

if (arr[4] == 10)
(prev)
10 == 10
found

// exact element match

Signature

K20-1052.

Date:

M	T	W	T	F	S	S

b) Linear < jump < binary

In the worst case jump search can have n/m jumps, if the last check element > than x (element we are searching). So (n/m + m-1) compalicons performed therefore O(√n) complexity.
Traversing backwards is easier but binary has complexity of O(log n), that is better!
Binary also uses Divide and conquer strategy but both searching needs sorted array.
Binary also needs more compaglisions and can not be efficient on linked lists.

b) Interpolation Search
It is an improvement for binary search, used in a situation where we have a sorted array with uniformly distributed values. It uses a formula to go different ~~arrg~~ array indexes.

$$p = low + \left\{ \frac{(hi - low)}{data[hi] - data[low]} \times (key - data[low]) \right\}$$

Exponential
Used for unbounded array, it finds the range where the data might be present then perform binary search on that range.

```
In while( i < n &&
    data[i] <= value)
    i = i * 2;

return BS(data, i/2,
        min(i, h-1), x);
```

low and high are

Page No.

RC

Signature

K20-1052.

Date:

M	T	W	T	F	S	S

using that formula it determine the position. It first checks on the position, if less than position go find new position on left else on right.

T.C = O(n)

upper and lower bound after which binary search is applied. It can outperform binary search if element at starting position.

T.C = (log i)

Signature

K20-1052.

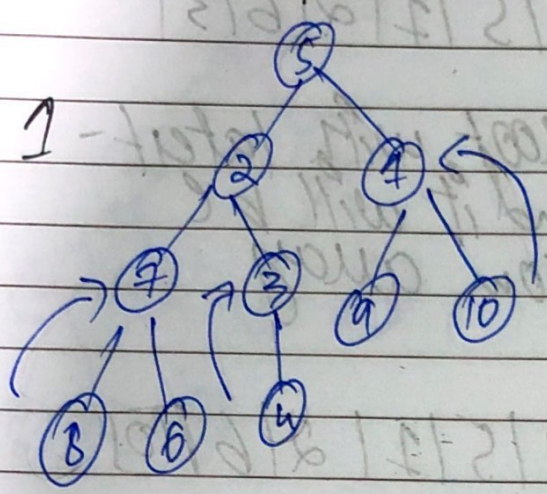
Date:

M	T	W	T	F	S	S

Q.6 Heap Sort $A[] = \{5, 2, 1, 7, 3, 9, 10, 8, 6, 4\}$

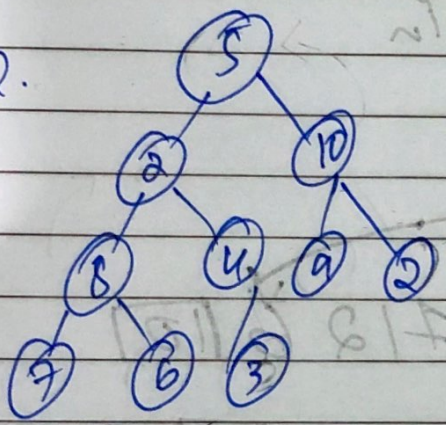
Building heap at first

1-



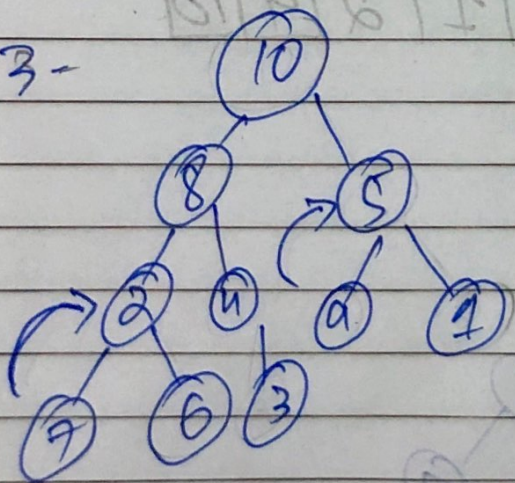
5	2	1	7	3	9	10	8	6	4

2.



5	2	10	8	4	9	1	7	6	3

3-

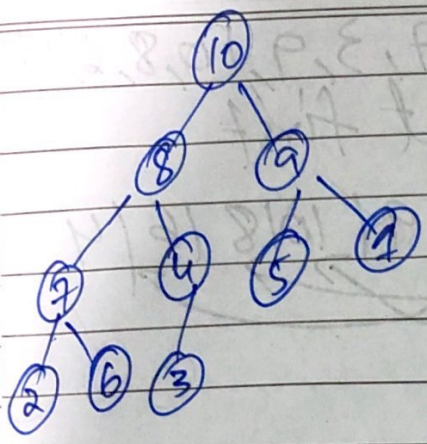


10	8	5	2	4	9	1	7	6	3

K20-1052.

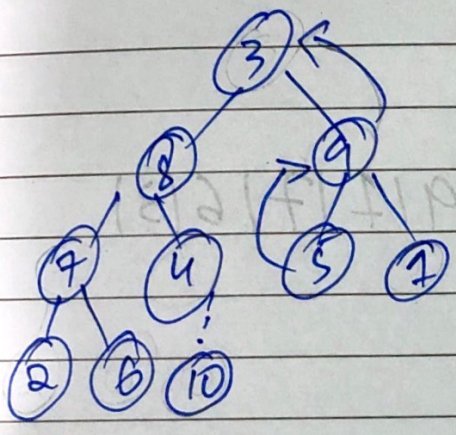
Date:

M	T	W	T	F	S	S



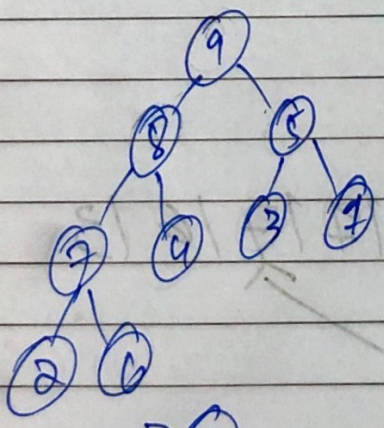
10	8	9	7	4	5	1	2	6	3

swap the root with latest element and it will be removed from array



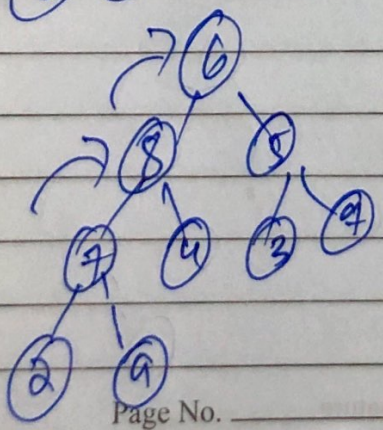
3	8	9	7	4	5	1	2	6	10

Heapify again

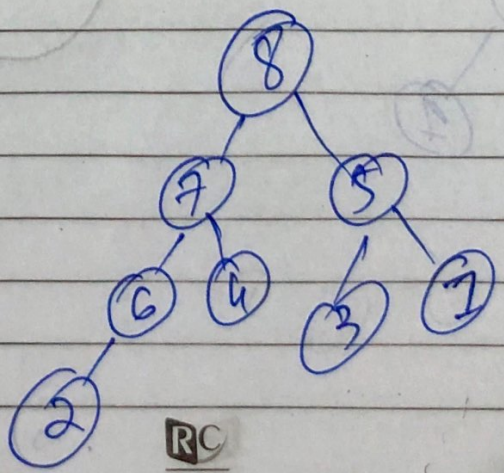


9	8	5	7	4	3	1	2	6	10

6	8	5	7	4	3	1	2	9	10



→

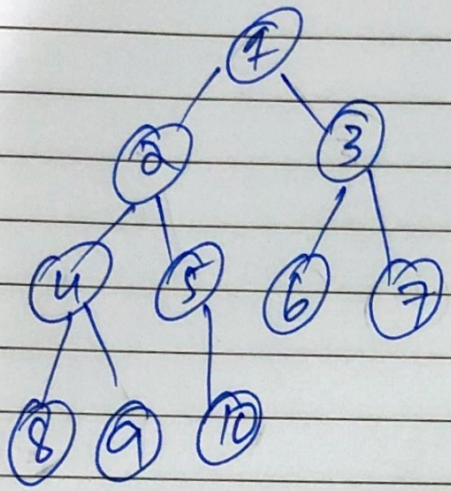


K20-1052.

Date:

M	T	W	T	F	S	S

In final iteration we have



1	2	3	4	5	6	7	8	9	10

RC

Signature