① Discuss the advantages and disadvantages of divide and conquer approach.

Advantages:

(i) It is a powerful tool for solving difficult problems

(ii) It often helps in the discovery of efficient algorithm.

(iii) Divide and conquer algorithms are naturally adapted for execution in multiprocessor machine, especially shared memory systems.

(iv) Divide and conquer algorithm naturally tend to make efficient use of memory caches.

Disadvantages:

Disadvantages of this paradigm is that if the instances obtained as a result of division are unbalanced then the divide and conquer are not very effective.

② Write the recursive and iterative binary search algorithm.

Recursive Binary Search

```
Algorithm Binsrch(a,i,l,x)
{
  if (l=i) then  // If small(P)
  {
    if (x=a[i]) then return i;
    else return 0;
  }
  else
  {  // Reduce p into a subproblem.
     mid=⌊(i+l)/2⌋;
     if (x=a[mid]) then return mid;
     else if (x< a[mid]) then
        return Binsrch(a,i,mid-1,x);
     else return Binsrch(a,mid+1,l,x);
  }
}
```

Iterative binary search

```
Algorithm Binsearch(a,n,x)
{
   low=1; high=n;
   while(low ≤ high) do
   {
     mid=⌊(low+high)/2⌋;
     if (x <a[mid]) then
          high = mid-1;
     else if (x > a[mid]) then
          low=mid+1;
     else return mid;
   }
     return 0;
}
```

(3) Write an general algorithm for divide and conquer with its time complexity.

```
Algorithm DAndC(P)
{
  if Small(P) then
      return S(P);
  else
  {
    divide p into smaller instances
       P1, P2, .... Pk , k ≥ 1;
    Apply DAndC to each of these
                    subproblems;
    return Combine (DAndC(P1),
                    DAndC(P2),
                      ⋮
                    DAndC(Pk));
  }
}
```

Time complexity

Recurrance of the form

$$T(n) = \begin{cases} T(1) & , n=1 \\ aT(n/b)+f(n), & n>1 \end{cases}$$

Consider $a=2$, $b=2$, $T(1)=2$ and $f(n)=n$

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

$= 2[2T(n/4)+n/2]+n$

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

$= 4\cdot[2T(n/8)+n/4]+2n$

$= 8T(n/8)+3n$

⋮

$= 2^k T\left(\frac{n}{2^k}\right) + kn$ [let, $n=2^k$]

$= nT\left(\frac{n}{n}\right) + n\log_2 n$

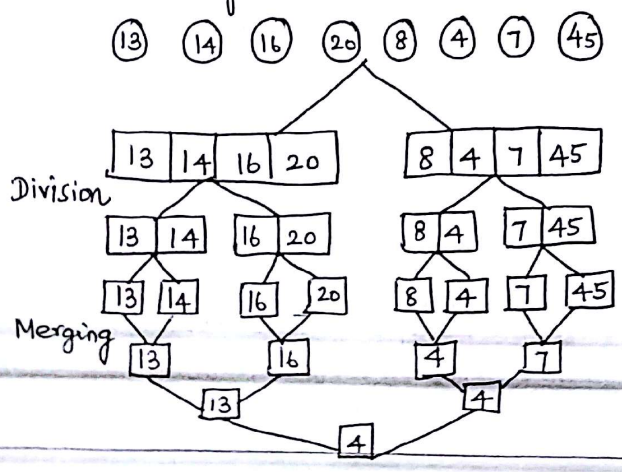
$T(n) = 2n + n\log_2 n$

$\therefore$ $\boxed{T(n) = n\log_2 n}$

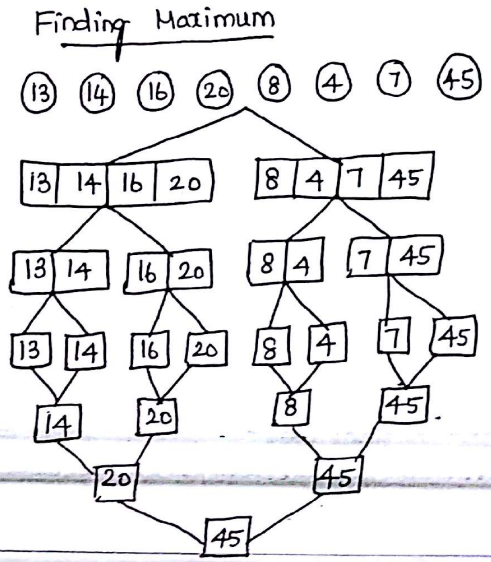
(4) Find the maximum and minimum elements of the following array using MAX-MIN algorithm $A = \{13, 14, 16, 20, 8, 4, 7, 45\}$ and also trace the recursive calls of Max-Min.

Solution:

Finding minimum



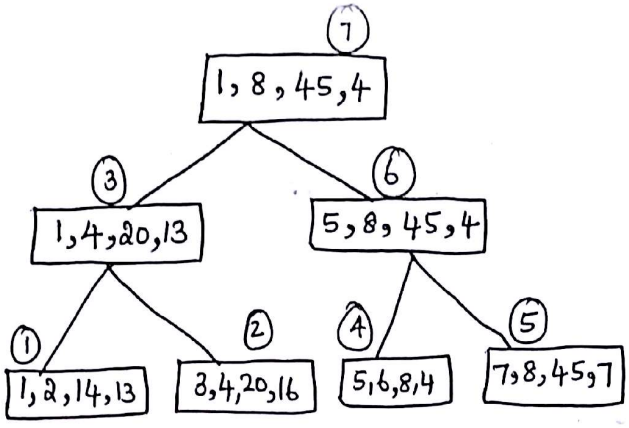
Finding Maximum



$A = \{ 13, 14, 16, 20, 8, 4, 7, 45 \}$
(1) (8)

$A = \{ 13, 14, 16, 20 \mid 8, 4, 7, 45 \}$
(1) (4) (5) (8)

$A = \{ 13, 14 \mid 16, 20 \mid 8, 4 \mid 7, 45 \}$
(1) (2) (3) (4) (5) (6) (7) (8)



⑤ Multiply the following two matrices using the Strassen's matrix method.

(i) $A = \begin{pmatrix} 2 & 5 \\ 5 & 2 \end{pmatrix} B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

(ii) $A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} B = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

Solution:

(i) $A = \begin{pmatrix} 2 & 5 \\ 5 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Here, $a_{11}=2,\ a_{12}=5,\ a_{21}=5,\ a_{22}=2$

$b_{11}=1,\ b_{12}=0,\ b_{21}=0,\ b_{22}=1$

Find,

$d_1 = (a_{11}+a_{22})(b_{11}+b_{22}) = (2+2)(1+1) = 8$

$d_2 = (a_{21}+a_{22})b_{11} = (5+2)(1) = 7$

$d_3 = a_{11}(b_{12}-b_{22}) = 2(0-1) = -2$

$d_4 = a_{22}(b_{21}-b_{11}) = 2(0-1) = -2$

$d_5 = (a_{11}+a_{12})b_{22} = (2+5)1 = 7$

$d_6 = (a_{21}-a_{11})(b_{11}+b_{12}) = (5-2)(1+0) = 3$

$d_7 = (a_{12}-a_{22})(b_{21}+b_{22}) = (5-2)(0+1) = 3$

Substitute these values,

$C_{11} = d_1+d_4-d_5+d_7 = 8-2-7+3 = 2$

$C_{12} = d_3+d_5 = -2+7 = 5$

$C_{21} = d_2+d_4 = 7-2 = 5$

$C_{22} = d_1+d_3-d_2+d_6 = 8-2-7+3 = 2$

∴ The resultant matrix is,

$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} 2 & 5 \\ 5 & 2 \end{pmatrix}$

Solution:

(ii) $A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

Here $a_{11}=1,\ a_{12}=2,\ a_{21}=2,\ a_{22}=1$

$b_{11}=2,\ b_{12}=1,\ b_{21}=1,\ b_{22}=2$

Find,

$d_1 = (a_{11}+a_{22})(b_{11}+b_{22}) = (1+1)(2+2) = 8$

$d_2 = (a_{21}+a_{22})\ b_{11} = (2+1)\,2 = 6$

$d_3 = a_{11}(b_{12}-b_{22}) = 1(1-2) = -1$

$d_4 = a_{22}(b_{21}-b_{11}) = 1(1-2) = -1$

$d_5 = (a_{11}+a_{12})\ b_{22} = (1+2)2 = 6$

$d_6 = (a_{21}-a_{11})(b_{11}+b_{12}) = (2-1)(2+1) = 3$

$d_7 = (a_{12}-a_{22})(b_{21}+b_{22}) = (2-1)(1+2) = 3$

Substitute these values,

$C_{11} = d_1+d_4-d_5+d_7 = 8-1-6+3 = 4$

$C_{12} = d_3+d_5 = -1+6 = 5$

$C_{21} = d_2+d_4 = 6+(-1) = 5$

$C_{22} = d_1+d_3-d_2+d_6 = 8-1-6+3 = 4$

∴ The resultant matrix is,

$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} 4 & 5 \\ 5 & 4 \end{pmatrix}$

(6) Discuss the Quick-hull algorithm.

Input: A set of n points, S

Output: Two closest point and distance

Begin

if n < threshold, then solve the problem by conventional algorithm.

else i=1, j=n

mid = $\lfloor \frac{(i+j)}{2} \rfloor$

d_l = Closestpair(S[1.... mid]);

d_r = closestpair (S[mid+1 ... n]);

d = min (d_l, d_r);

end if

for index = 1 to n do

if ((S[index] >= S(mid).x − d) or (S[index] <= S(mid+1).x + d)) then

Append S[i] to array V

end if

end for

Sort list v based on y-coordinates

Let d_{across} = minimum of distance among six points of array v

return (min (d_{min}, d_{across}))

end.

① Explain in detail about binary search algorithm and derive its time complexity.

Binary search:

Binary search is used to search a given 'key' in an array of numbers. Binary search is more advantageous in comparison to linear search, as it is faster and easier to implement.

Informally, the binary search algorithm can be written as,

Step1: Read the ordered array of elements and search for a target key.

Step2: Find the middle element of ordered array.

Step3: Compare between the key and the middle element

3a: If key = middle element then display the message key is found. stop.

3b: If key < middle element then search the lower half of the ordered array A [1 .... middle-1].

3c: If key > middle element then search the upper half of the ordered array A [middle+1 .... n].

Formal algorithm

Recursive Binary search

```
Algorithm Binsrch (a,i,l,x)
{
  if (l=i) then
  {
    if (x=a[i]) then
        return i;
    else
        return 0;
  }
  else
  {
     mid = ⌊(i+l)/2⌋;
     if (x=a[mid]) then return mid;
    else if (x < a[mid]) then
      return Binsrch(a, i, mid-1, x);
    else
      return Binsrch(a, mid+1, l, x);
  }
}
```

Iterative Binary search

```
Algorithm Binsearch (a,n,x)
{
   low = l ; high = n;
   while ( low ≤ high) do
   {
     mid = ⌊(low+high)/2⌋;
     if (x<a[mid]) then
         high = mid -1;
     else if (x>a[mid]) then
         low = mid+1;
     else
         return mid;
   }
   return 0;
}
```

Example: Let us select the 14 entries.

$-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151 \Rightarrow a[1:14]$

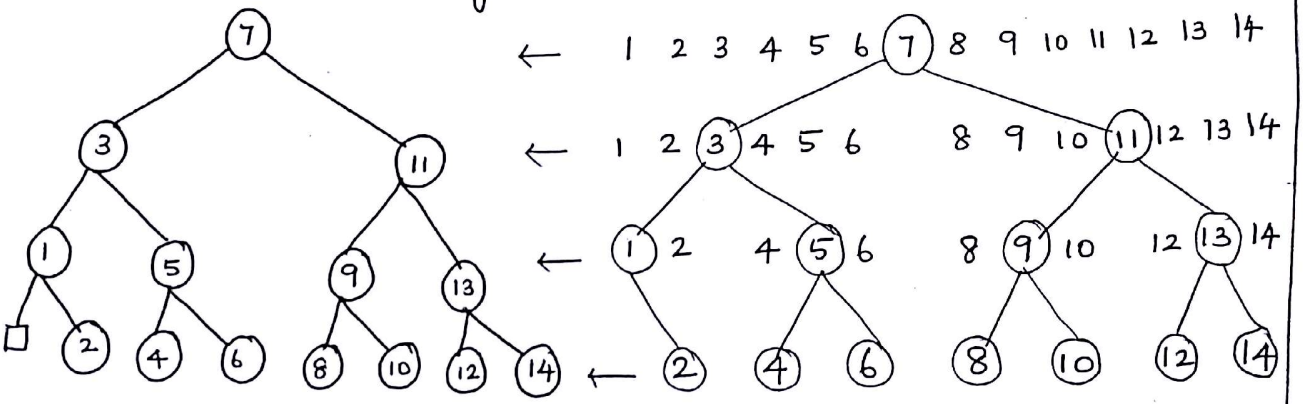
[1] [2] [3] [4] [5] (6) [7] [8] [9] [10] [11] [12] [13] [14]

Search $x = 151$

low	high	mid	
1	14	$\lfloor \frac{1+14}{2} \rfloor = 7$	→ Check $a[7] = 54 < x = 151$
8	14	$\lfloor \frac{8+14}{2} \rfloor = 11$	→ low = mid+1, check $a[11] = 125 < x = 151$ (7+1)
12	14	$\lfloor \frac{12+14}{2} \rfloor = 13$	→ low = mid+1, check $a[13] = 142 < x = 151$ (11+1)
14	14	$\lfloor \frac{14+14}{2} \rfloor = 14$	→ low = mid+1, check $a[14] = 151 = x$ (13+1)

key found ($x = 151$) in location 14.

Binary decision tree for binary search, $n = 14$ is,



Time complexity:

Worst_case analysis $T(n) = \log_2 n$

Best_Case analysis $T(n) = O(1)$

Average case analysis $T(n) = O(\log n)$

} ⇒ Successful searches.

Unsuccessful Searches → Best, Average, worst } $= \Theta(\log n)$.

② Explain in detail about Quick sort algorithm and derive its time complexity. (partitioning algorithms).

Quicksort

It uses the divide and conquer strategy for sorting elements of an array. It uses a pivot element to divide the array into two parts.

Quick sort has two stages

(i) The array is divided using a pivot element.

Let the pivot element be v. An array A is partitioned into two arrays such that $array1 = \{x \in A - \{v\} \mid x \leq v\}$ and $array2 = \{x \in A - \{v\} \mid x \geq v\}$.

(ii) Sort the array1 and array2 and combine.

Informally,

Step1: pick an element of the array as the pivot element V.

Step2: partition the array using pivot element V.

Subarray1 ≤ pivot element V. (left side)

Subarray2 ≥ pivot element V. (Right side)

Step3: Sort Subarray1 and Subarray2 recussively.

Step4: Combine Subarray1 and Subarray2. ⇒ sorted array.

Algorithm

```
Algorithm QuickSort(P,q)
{
  if (P<q) then
  {      // divide P into two subproblems
    j = Partition(a, P, q+1);
        // Solve the subproblems
    QuickSort(P, j-1);
    QuickSort(j+1, q);
  }
}
```

```
Algorithm partition(a, m, P)
{
  v = a[m]; i = m; j = P;
  repeat
  {
    repeat
      i = i+1;
    until (a[i] ≥ v);          } Subarray1
    repeat
      j = j-1;
    until (a[j] ≤ v);          } Subarray2
    if (i<j) then Interchange(a,i,j);
  } until (i ≥ j);
  a[m] = a[j]; a[j] = v; return j;
}
```

Pivot

```
Algorithm Interchange(a,i,j)
{
  P = a[i]; a[i] = a[j]; a[j] = P;
}
```

Time complexity:

Best case Complexity analysis ⇒ partition element (pivot) is placed exactly in the middle of the array.

Recurrence equation $T(n) = 2T(n/2) + n$

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

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

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

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

$= 4[2T(n/8) + n/4] + 2n$

$= 8T(n/8) + n + 2n$

$= 8T(n/8) + 3n$

$\vdots$

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

Assume $2^k = n$

$\therefore \log 2^k = \log n$

$k \log 2 = \log n$

$k = \log n$

$\therefore T(n) = nT(n/n) + \log n \cdot (n)$

$= nT(1) + n\log n$

$= n + n\log n$

$\boxed{T(n) = \theta(n \log n)}$

Worst case Complexity analysis ⇒ partition element (pivot) is placed exactly first element of the array.

$\therefore T(n) = 1 + 2 + 3 + \cdots + n$

$= \frac{n(n+1)}{2}$

$= \frac{n^2}{2} + \frac{n}{2}$

$\boxed{T(n) = \theta(n^2)}$

Average case complexity analysis ⇒ Any element can be a pivot element.

Let $C(n)$ ⇒ No. of comparisons required for sorting the no. of an array $A[1 \ldots\ldots n]$.

$C_A(n)$ ⇒ Average case analysis.

Two subarrays to be sorted are $a[m:j]$ and $a[j+1 : p-1]$ with probability $1/(p-m)$, $m \le j < p$.

From this we obtain the recurrence,

$$C_A(n) = n+1 + \frac{1}{n} \sum_{1 \le k \le n} [C_A(k-1) + C_A(n-k)] \quad \text{——} (1)$$

Here $n+1$ ⇒ No. of element comparisons required by partition on its 1st call.

Initial condition $C_A(0) = C_A(1) = 0$.

$$C_A(n) = n+1 + \frac{1}{n} 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right] \quad \text{——} (2)$$

Multiplying both sides by n, we obtain

(2) ⇒ $$n C_A(n) = n(n+1) + \frac{n}{n} 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right]$$

$$n C_A(n) = n(n+1) + 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right] \quad \text{——} (3)$$

Replacing n by $n-1$

(3) ⇒ $$(n-1) C_A(n-1) = (n-1)(n-1+1) + 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-2)\right]$$

$$(n-1) C_A(n-1) = n(n-1) + 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-2)\right] \quad \text{——} (4)$$

Subtract (4) from (3)

$$n C_A(n) - (n-1) C_A(n-1) = n(n+1) - [n(n-1)] + 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-1)\right] - 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-2)\right]$$

$$n C_A(n) - (n-1) C_A(n-1) = n^2 + n - n^2 + n + 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-2) + C_A(n-1)\right] - 2\left[C_A(0) + C_A(1) + \cdots + C_A(n-2)\right]$$

$$n C_A(n) - (n-1) C_A(n-1) = (n+1) + 2C_A(n-1) \quad \text{——} (5)$$

Simplify equ (5)

$$nC_A(n) - (n-1)C_A(n-1) = 2n + 2C_A(n-1)$$

$$nC_A(n) = 2n + 2C_A(n-1) + (n-1)C_A(n-1)$$

$$= 2n + C_A(n-1)\left[2 + n - 1\right]$$

$$nC_A(n) = 2n + C_A(n-1)(1+n)$$

Assume $\boxed{n = \frac{1}{n+1} \Rightarrow n+1 = \frac{1}{n}}$

$$\frac{C_A(n)}{n+1} = \frac{2}{n+1} + \frac{C_A(n-1)}{n}$$

$\therefore$

$$\frac{C_A(n)}{n+1} = \frac{C_A(n-1)}{n} + \frac{2}{n+1}$$

Recursively apply this,

$$\frac{C_A(n)}{n+1} = \frac{C_A(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \frac{C_A(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$\vdots$$

$$= \frac{C_A(1)}{2} + 2 \sum_{3<k\leq n+1} 1/k$$

$$= \frac{0}{2} + 2 \sum_{3<k\leq n+1} 1/k$$

$$\boxed{\frac{C_A(n)}{n+1} = 2 \sum_{3<k\leq n+1} 1/k} \quad \text{——— (6)}$$

Since $\sum_{3\leq k\leq n+1} 1/k \leq \int_{2}^{n+1} \frac{1}{x}\,dx = \log_e(n+1) - \log_e 2$ [probability density function]

(6) $\Rightarrow \frac{C_A(n)}{n+1} \leq 2\left[\log_e(n+1) - \log_e 2\right]$

$$C_A(n) \leq 2(n+1)\left[\log_e(n+1) - \log_e 2\right]$$

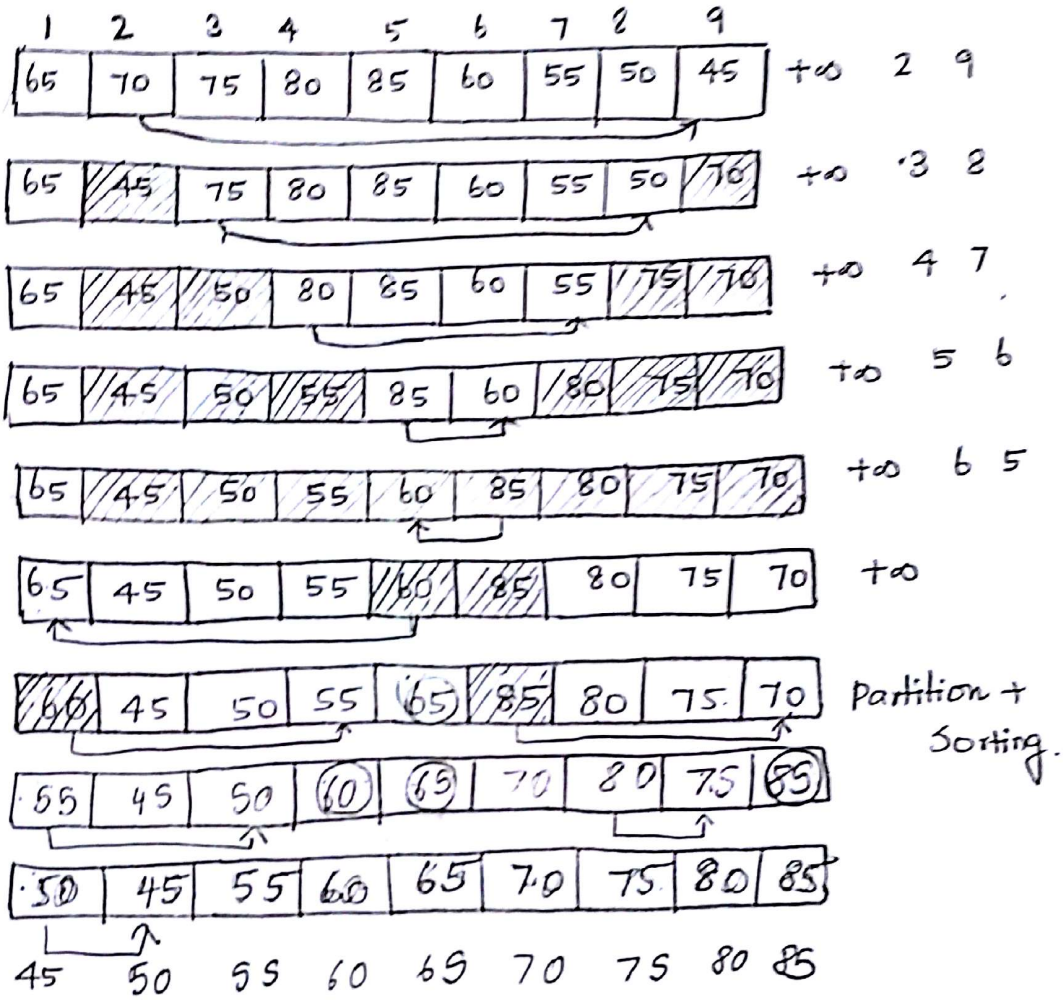
$$C_A(n) \leq 2(n+1)\log_e(n+1) - 2(n+1)\log_e 2$$

$\rightarrow \boxed{C_A(n) = O(n \log n)}$

Quick Sort: Example: Consider the array of elements

$A = \{65, 70, 75, 80, 85, 60, 55, 50, 45\}$

Take. Pivot Element = 65 (First element)



③ Explain in detail about merge sort algorithm and derive its time complexity.

Merge sort:

⇒ Merge sort uses the divide-and-conquer strategy for sorting unordered arrays.

⇒ the strategy used for merge sort involves the stages of the divide and conquer paradigm.

1st phase (Divide) ⇒ Divide the array of numbers into two equal parts. If necessary, these subarrays are divided further.

2nd phase (Conquer) ⇒ Sorting of subarrays recursively and combine the sorted subarrays to give a final sorted list.

Informally, the procedure for mergesort is,

Step1: Divide an array A into subarrays B and C.

Step2: Sort subarray B recursively; this yields B sorted subarray.

Step3: Sort subarray C recursively; this yields C sorted subarray.

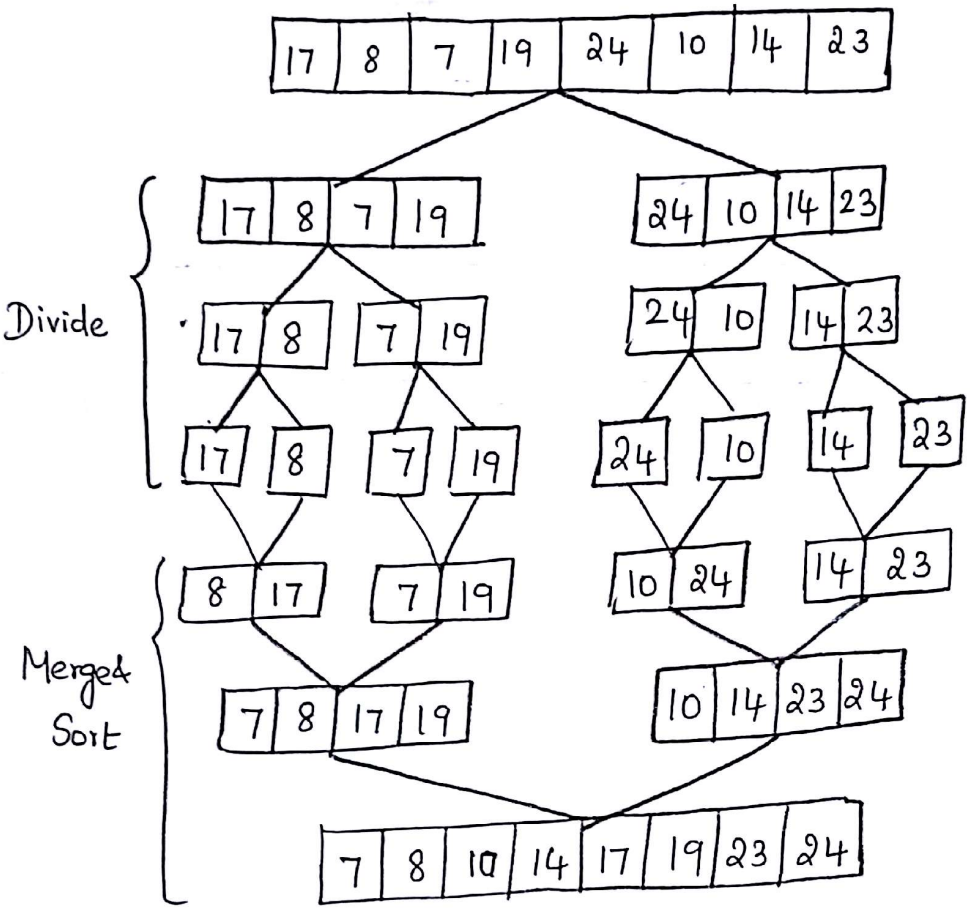
Step4: Combine B and C sorted subarrays to obtain the final sorted array A.

Algorithm:

```
Algorithm mergesort(A [first ... last])
Input: Unsorted array A with first=1 and last=n
Output: Sorted array A.
✓Begin
  [if(first==last) then
     return A[first]
  else
     mid = ⌊(first+last)/2⌋
  [for i ∈ {1, 2, ..., mid}
       B[i] = A[i]
  End for
  [for i ∈ {mid+1, ... n}
       C[i] = A[i]
  End for
  // Conquer and merge
  mergesort(B[1 ... mid])
  mergesort(C[mid+1 ... n])
  merge(B, C, A)
✓End.
```

```
Algorithm merge (B,C,A)
Input: Two Sorted arrays B and C
output: Sorted array A.
Begin
    i=1  // Pointer i tracks array B
    j=1  // pointer j tracks array C
    k=1  // pointer k tracks array A
    m=length(B)  // Array B is of size m
    n=length(C)  // Array C is of size n.
  While( (i<=m) and (j<=n)) do
      if (B[i] < C[j]) then
            A[k]=B[i]
              i=i+1
      else
          A[k]=C[j]
            j=j+1
      End if
            k=k+1
  end while
    if (i>m) then
      while(k<=m+n) do
          A[k]=C[j]
          j,k=j+1, k+1
      end while
    else if (j>n) then
        while (k<=m+n) do
            A[k]=B[i]
            i, k=i+1, k+1
        end while
    end if
        return (A)
End.
```

Example: Consider the array A = { 17, 8, 7, 19, 24, 10, 14, 23 }



Complexity:

Recurrence relation for merge sort operation is,

$$T(n) = \begin{cases} a & , n=1 \\ 2T(n/2)+Cn & , n>1 \end{cases}$$

Here a, C => Constant.

By using substitution,

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

$$= 2\left(2T(n/4) + Cn/2\right) + Cn$$

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

$$\vdots$$

$$= 2^k T(1) + kCn$$

$$T(n) = an + Cn\log n$$

$\Leftarrow$ [$n = 2^k$, $\log n = \log 2^k$, $\log n = k \log_2^2$, $k = \log n$]

$\therefore$ Complexity $\boxed{T(n) = O(n \log n)}$

(4) Explain in detail about finding maximum and minimum elements (Max-Min) and derive its time complexity. [Example Refer: 4th Question]

Concept:

⇒ Divide and conquer strategy is used to find the minimum and maximum elements of an array effectively.

⇒ The idea is to divide the given array into subarrays and find recursively the maximum and minimum elements of the subarrays.

⇒ Then the global maximum and minimum of an array can be found by comparing the maximum and minimum elements of the subarrays.

Procedure:

Step1: Recursively divide an array A into subarrays.

Step2: Recursively find the maximum and minimum elements of the subarrays.

Step3: compare the maximum/minimum elements of the sub arrays to get the global maximum and minimum of the given array.

Algorithm:

```
Algorithm MaxMin(i, j, max, min)
{
    if (i=j) then max=min=a[i];
  else if (i=j-1) then
  {
    if (a[i]<a[j]) then
    {
      max=a[j]; min=a[i];
    }
    else
    {
      max=a[i]; min=a[i];
    }
  }
  else
  {
      mid=⌊(i+j)/2⌋;
    // Solve the subproblems
      MaxMin(i, mid, max, min);
      MaxMin(mid+1, j, max1, min1);
    // Combine the solutions
      if (max<max1) then max=max1;
} }   if (min>min1) then min=min1;
```

Time complexity

A conventional maximum/minimum requires 2(n-1) comparisons for best, average & worst cases.

By using diride and conquer, the recurrence equation is,

$$T(n) = \begin{cases} 2T(n/2)+2 & n>2 \\ 1 & n=2 \\ 0 & n=1 \end{cases}$$

Assume $n = 2^k$, using substition method

$$T(n) = 2T(n/2) + 2$$
$$= 2\left(2T(n/4)+2\right)+2$$
$$= 4T(n/4) + 4 + 2$$
$$\vdots$$
$$= 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i$$
$$= 2^{k-1}(1) + 2^k - 2$$
$$= \frac{2^k}{2} + 2^k - 2$$
$$= \frac{n}{2} + n - 2.$$

$$\boxed{T(n) = \left(\frac{3n}{2}\right) - 2}$$

(5) Explain in detail about Closest-pair algorithm using divide and conquer approach. [Refer pageNo: 291 – 293]

(6) Explain the following convex hull.

a) Quick hull [Pg: 293 - 295] ⇒ Concept + Algorithm (steps) + Complexity analysis

b) Merge hull [Pg: 295 - 296] ⇒ Concept + Algorithm (steps) + Complexity analysis.