A. C. Patil College of Engineering Kharghar Navi-Mumbai Maharashtra

Name : Priyush Bhimrao Khobragade

Roll NO: 52
PRN NO :211112018

Subject : Analysis of Algorithms (AOA)
Assignment-02

Page No.
Date

Assignment No-02

Q.1. Write a program for finding the smallest and largest element from an array using divide & conquer approach & detile the its complexity.

```
#include <stdio.h>
int DAC_MAX(int a[], int index, int l);
int DAC_MAX(int a[], int index, int l);
int DAC_MAX(int a[], int index, int l);
{
  int max;
  if (index >= l-2)
  {
    if (a[index] > a[index+1])
      return a[index];
    else
      return a[index+1];
  }
  MAX = DAC_MAX(a, index+l, i);
  if (a[index] > max)
    return a[index];
  else
    return max;
}
int DAC_min (int a[], int index, int l)
{
  int min;
  if (index >= l-2)
  {
    if [a[index] < a[index+l]]
```

```
return a[index];
else
 return a[index+1];
}
 min = DAC_Min (a, index+1, I);
  if (a[index] < min)
 return a[index];
 else
  return min;
 }
 int main()
{
 int min, max, M;
 int a[7] = {70, 250, 50, 80, 140, 12, 14};
 max = DAC_Max(a, 0, 7);
 min = DAC_min(a, 0, 1);
 printf("The minimum number in given array is: %d\n", min);
 printf("The maximum number in given array is: %d\n", max);
 return 0;
}
```

Output :-

The minimum number in given array is = 12

The maximum num in given array is = 250

- Complexity :-

If T(n) represent this no; then recurring reccurance relation is $T(n) = T([n/2] + T(n/2) + 2]$ $n > 2$

1 $n = 2$

1 $n = 1$

When n is a power of 2 $n = 2k$ for some positive integer then

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

$$\Rightarrow \boxed{T(n) = 3n/2 - 2}$$

Q.2) Write a program for the Tower of Hanoi puzzole.

```
#include <stdio.h>
void TOH (int n, char x, char y, char z)
{
  if (n>0)
{ TOH (n-1, x, z, y);
  printf ("\n%c to %c", x, y);
  TOH (n-1, z, y, x);
  }
}
int main()
{
  int n=3
  TOH (n, 'A', 'B', 'C');
}
```

Output :-

A to B
A to C
B to C
A to B
C to A
C to B
A to B.

Page No.
Date

Q.3 Explain merge sort with example and comment on its complexity

- Merge sort is one of the most efficient sorting algorithm.
- it works on the principle of divide & conquer based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element & merging those sublist in a manner that result into a sorted list.

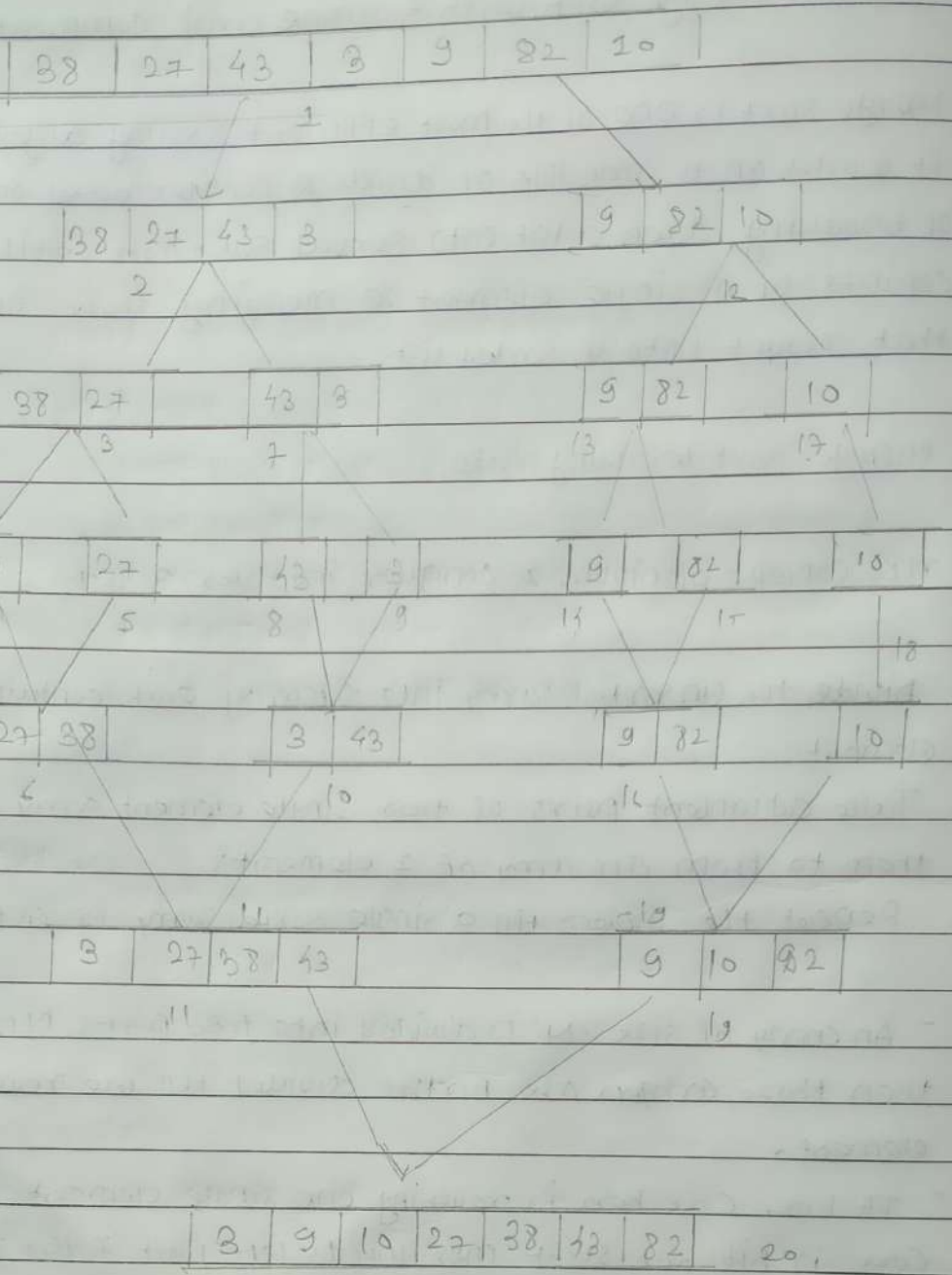
· Merge-sort working rule:

· The concept of divide & conquer involves 3 steps:-

1) Divide the unsorted array into subarray each containing a single element.
2) Take adjacent pairs of two single element array and merge them to from an array of 2 elements.
3) Repeat the process till a single sorted array is obtained.

- An array of size 'N' is divided into two parts N/2 size of each then those arrays are further divided till we reach a single element.
- The base case here is reaching one single element. when the base case is hit, we start merging to left part & the right part & we get a sorted array at the end.
- Merge sort repeatedly until ~~each~~ break down an array into several subarrays until each subarrays consists of a single element and merging those subarrays in a manner that result in a sorted array.

Example of Merge sort:



then no indicate in which step it is done.

* Algorithm for merge sort :-

Step 01 :- Find the middle index to carry middle = (first + last)/2

Step 02 :- Divide the array from the middle.

Step 03 :- Call merge sort for the first half of the array.
Merge sort (array, first, middle.

Page No.
Date

Step 04 :- Call Merge Sort for the second half of the array merge Sort (array, middle+1, last)

Step 05 :- Merge the two sorted halves into a single sorted array

* Program on Merge sort :-

```
#include <studio.h>
# define = max to
int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0};
int b[10];
void merging (int low, int mid+1, i=low; 1 <= mid && le <= high;
                i++);
{
  if (a[12] =< a a[12])
  b[i] = a [i++];
  else
  b[i++] = a[]++];
  while (12 <= high)
  b [i++] = a [12++];
  for (i = low ; i <= high; i++)
  a[i] = b[i];
  }
  void sort (int low, int high)
{  int mid.
  if (low < high)
{ mid = (low+high) /2 ;
  Sort = (low, mid);
  Sort ( mid+1, high);
```

```
        merging (low, mid, high);
    }
    else
    {
    } return;
    }
    int main()
    {
    int i;
    printf("list before sorting\n");
    for(i=0; i<=max; i++)
    {
      printf("%d", a[i]);
    }
    Sort(0, max);
    printf("\n list after sorting\n");
    for(i=0; i<=max; i++)
    {
      printf("%d", a[i]);
    }
```

Output :-

list before Sorting

10 14 19 26 27 31 33 35 42 44 0

List after sorting:

0 10 14 19 26 27 31 33 35 42 44

- Complexity of Merge Sort :-

The running time of merge sort in the given average case & the worst case can be given as O(nlogn). Best time complexity is also same as worst & average time complexity.

Q.4) Explain the quick sort with example & comment on its complexity.

- Like merge sort, quick sort is a divide and conquer algorithm it picks an element as pivot and partitions the given array around the picked pivot.
- There are many different version of quicksort that pick pivot in different ways.

1) Always pick first element as pivot.
2) Always pick last element as pivot. (implemented below)
3) pick a random element as pivot
4) pick median as pivot.

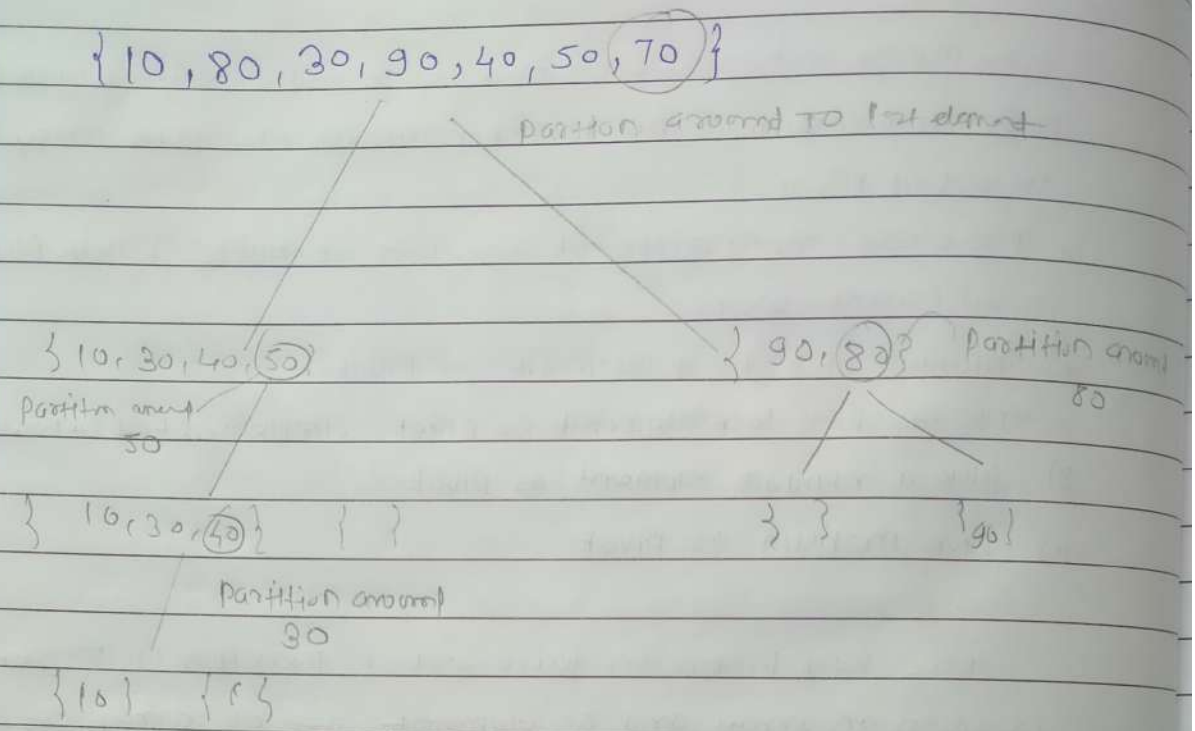
- The key process in quicksort is partition(). Target of partitions is given an array and an element ~~not~~ of array as pivot, put x at its correct position in sorted array & put all smaller element (smaller than x) before x, and put all greater element (greater than x) after x. all this should be done in linear time.

- The logic is simple, we start from the left most element and keep track of index of smaller (or equal to) element as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

- **Complexity of quicksort**

To sort an array of n distinct element, quicksort takes O(nlogn) time in expectations averaged all over n! permutations of n element with equal probability.

* Example of quick sort:-



* Algorithm for quick sort:-

Step 01:- make the right most index value pivot.
Step 02:- partition the array using pivot value
Step 03:- quicksort left partition recursively
Step 04:- quicksort right partion recursively.

Page No.
Date

• Program code on quick sort :-

```
#include <studio.h>
void quicksort (int[], int, int);
int main ()
{
int list [50];
int size i;
printf (" Entee the no of element ": );
Scanf ("%d", & size];
printf ("%d Entee te element to co, stored \n");
for (i=0; i<size; i++)
    {
      Scanf ("%d", & list [i]);
    }
    quicksort (list, 0, size - 1);
    Print f (" After carrying quick sort \n");
    for (i=0; i<size; i++) :
      {
        printf ("%d", list[i]);
      }
    Print ("\n");
    return 0;
  }
void quicksort (int list[] int low, int high)
  {
    int pivot, i, j, temp;
  if (low < high)
}
    pivot = low;
```

```
i=low;
j=high;
while(i<j):
 {
  while(list[i]<=list[pivot]&&i<=high)
    {
      i++;
  }while(list[i]>list[pivot]&&j>low)
    {
      j--;
  }
   if(i<j)
    {
     temp=list[i];
     list[i]=list[j];
     list[j]=temp;
     }
 }
 temp=list[j];
 list[j]=list[pivot];
 list[pivot]=temp;
 quicksort(list,low,j-1);
 quicksort(list,j+1,high);
}
}
```

Output:-

Enter the number of the element :- 6

Enter the element to be sorted :-

67
45
24
98
12
38

After applying quicksort :

12, 24 38 45 67 98

Q.5] Write the an algorithm for Binary search method also mention its Best case, worst case & average case complexity.

→

- Algorithm for Binary search method

Step 01 :- Read the search method from the user

Step 02 :- Find the middle element in the sorted list

Step 03 :- Compare the search element with the middle element in the sorted list

Step 04 :- If the both are matched, then check whether the search element is smaller or larger than the middle element.

Step 05 :- If the element is smaller than middle element, repeat Step 1, 2, 3 & 04 for the right sublist of the middle element.

Step 06 :- If the search element is large than middle element, repeat step 2, 3, 4 & 5 for the right sublist of the middle element.

Step 07 :- Repeat the same process until we find search element in the list or until sublist content only 1 element.

Step 08 :- If that element doesn't match with the search element, than display "element is not found in the list" & terminate the function.

* Complexity

1) Best case for Binary search :- O(1)
2) Worst case for Binary search :- O(1)
3) Average case for Binary search :- O(log n).