Assignment-6

K. Mounika
AP19110010101
CSE-H

1) Take the elements from the user and sort them in descending order and do the following

a. using Binary search find the element and the location in the Array where the element is asked from user.

b. Ask the user to enter any 2 locations Print the sum and product of values those locations in the sorted array.

```
# include <stdlib.h>
# include <stdio.h>
  int comparator (const void* a, const void *b){
{ return (*(int*)b - *(int*)a);
 }
int binary search (int arr[], int size, int search){
    int beg=0, end = size-1, mid;
    while (beg<= end){
        mid = (beg+end)/2
        if(arr[mid] == search){
            return mid;
         }
       else if (arr[mid]< search){
            end = mid-1;
       }
        else beg= mid+1;
   }
   return -1;
```

Scanned with CamScanner

4

```
int main()
{
  int arr[100], size, search, i, Pos=-1, loc1, loc2;
  Printf("\n enter the size of the array min:100");
   scanf("%d", &size);
  Printf("\n enter the elements in array\n");
    for(i=0; i<size, i++){
       scanf("%d", &arr[i]);
  }
 qsort(arr, size, size of(int), comparator);
   Printf("\n The sorted Array is:\n");
     for(i=0; i<size; i++)
     {
       Printf("%d", arr[i]);
     }
     Printf("\n enter search element");
     scanf("%d", &search);
      Pos = binary search(arr, size, search);
      if(Pos==-1)
        Printf("Not found\n");
      else:
        Printf("\n The %d search element is
        found at inder %d\n", search, Pos);
    Printf("enter two indexes\n");
    scanf("%d %d", &loc1 &loc2);
    Printf("sum is %d\n", arr[loc1]+arr[loc2]);
    Printf("Product is %d\n", arr[loc1]*arr[loc2]);
```

Scanned with CamScanner

output:

```
Enter the size of the Array (max 100) 4
Enter elements in array
1 2 3 4
The sorted Array is :
4 3 2 1
Enter search element 2
The 2 search element is found at index 2
Enter two indexes
1 2
sum is 3
Product is 2
```

2) Sort the array using Merge sort where elements are taken from the user and find the Product of kth elements from first and last where k is taken from the user.

```
# include <stdio.h>
# Define MS 100
int x [MS];
void merge (int m1, int m1, int m2, int m2)
{
    int i, j, k, temp [MS];
    k=0;
    i = m1
    j = m2;
```

Scanned with CamScanner

```
while (i<= n1) && (j<= n2)) {
if (x[i] < x[j]) {
  temp[k]= x[i]; i++; k++;
}
else {
   temp[k] = x[j]; i++; k++;
  }
}
while (i<= n1){
   temp[k] = x[i]; j++ i k++;
}
for (i= m1, k=0; i<n2; i++, k++){
      x[i]= temp[k];
}}
Void merge sort (int *my, int ny){
  if (my<ny)
  {int mid = (my+ny)/2;
   merge sort (my, mid);
   merge sort (mid+1, ny);
   merge (my+mid, mid+1, m+y);
}}
int main (){
  int i, num, Product=1;
  Printf("\n enter the size of the Array max(100):");
  Scanf ("%d", &num);
```

Scanned with CamScanner

```
for (i=0; i<n; i++){
  Printf ("x [%d] | t = "; i);
  scanf ("%d", &x[i]);
}
  Printf ("\n The Product till the kth elements is
  return 0;                                    % d \n", Product);
}
```

Enter the size of the Array 5

x@[0] = 2

x[1] = 6

x[2] = 8

x[3] = 10

x[4] = 20

enter k

1

The product till kth element is 60

3) Discuss insertion sort and selection sort with examples.

Insertion sort:

Insertion sort works by inserting the set of values in the existing sorted file. It constructs the sorted array by inserting a single element at a time. This process continues until which array is sorted in same order.

Scanned with CamScanner

The primary concept behind insertion sort is to insert each item into its appropraite place in the final list. The insertion sort method saves an effective amount of memory.

Working of Insertion sort:-

1) It uses two sets of array where one stores the sorted data and other on unsorted data.
2) The sorting algorithm works until there are elements in unsorted sets
3) Let's assume there are 'n' number elements in the Array. Initially, the element with index 0 (B=0) exists in sorted set. Remaining elements are in the unsorted position of the list.
* the first element of unsorted partion has Array with Index 1 (If (B=0)).
* After each in insertion, it chooses the first element of the insorted partition and inserts it into the proper sorted set.

Complexity of Insertion sort:-

The best case Complexity of insertion sort is O(n) times i.e, when the Array is Previously sorted. In the same way, when the Array is sorted in reverse order, the first element of the unsorted array is to compared with each element in the sorted set. So, in the worst case the running time of insertion sort is Quadratic i.e, $O(n^2)$ In Average case also it has to make the minimum $(K-1)$ / 2 Comparisons. Hence, the Average case also has

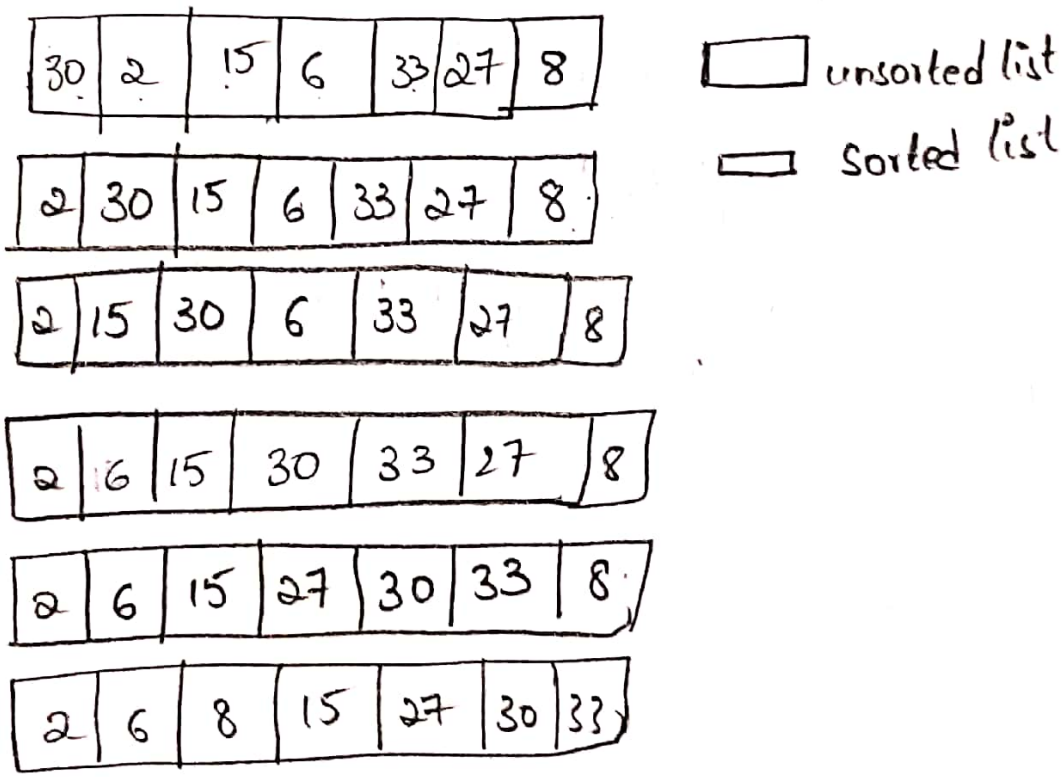
Scanned with CamScanner

running time O(n²)

Advantages of insertion sort

* the additional memory space required of insertion sort is less (i.e, O(1))
* Easily implemented and very efficient when used with small sets (or) data.
* It is considered to be live sorting technique as the list can be sorted as the new elements are recieved
* It is faster than other sorting Algorithm.

Example:-



Selection sort:-

The selection sort Perform sorting by searching for the minimum value number and placing it into the first (or) last position according to the order (ascending (or) descending). the Process of searching minimum key and placing it in the Proper

Scanned with CamScanner

Position is continued until the all elements are placed at right position.

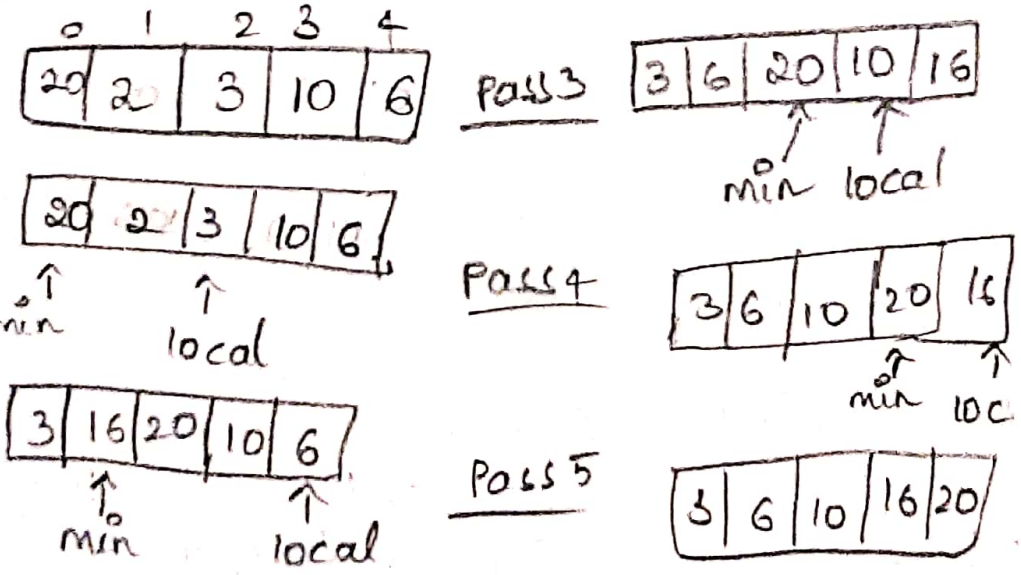
Working of selection sort :-

suppose an array ARR with N elements in the memory. In the first pass, the smallest Key is searched along with its position of ARR[Pos] is swapped with ARR[0]. Therefore ARR[0] is sorted. In the second pass, again the position of the smallest value is determined in the sub-array of N-1 elements interchange the ARR[Pos] with ARR[1] In the pass N-1, the same process is performed to sort the N number of elements.

Advantages of selection sort :-

* It performs well on a small list.
* It is an place sorting algorithm, no additional temporary storage is required beyond what is needed to hold the original list.

Example :-



Scanned with CamScanner

Complexity of selection sort :-

As the working of selection, sort does not depend on the original order of the elements in the array, so there is not much difference between best case and worst case complexity of selection sort.

The selection sort selects the minimum value. element in the selection process. All the 'n' numbers of elements are scanned, therefore n-1 comparison are made in the first pass. Then the elements are interchanged. Similarly in second pass also find the second smallest element we require scanning of rest n-1 elements and the process is continued till the whole array sorted.

Running time complexity of selection sort is $O(n^2)$

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

4) sort the array using bubble sort where elements are taken from the user and display the elements.
(i) In Alternate order
(ii) sum of elements in odd positions and Product of elements in even positions
(iii) Elements which are divisible by m where m is taken from user.

```
# include <stdio.h>
void main()
{ int a[100], num, x, y, temp, s=0, prod=1, k, m;
  Printf("enter number of elements to be entered:");
  scanf("%d", &num);
  for (x=0; x<num; x++) Printf("enter %d integers\n", num);
  {
```

Scanned with CamScanner

```
scanf("%d", &H[i]);
}
for (x=0; x<num-1; x++)
{
  if (H[y] > H[y+1])
  {
    temp = H[y];
    H[y] = H[y+1];
    H[y+1] = temp;
  }
}
}
Printf("\n Sorted list in Ascending order:\n");
for (x=0; x<num; x++)
{
  Printf("%d\n", H[x]);
}
Printf("\n Alternate order");
  for (x=0; x<num; x++)
  {
    if (x%2 ==0)
    {
      Printf("%d", H[i]);
    }
  }
  for (x=0; x<n; x++)
  {
    if (x%2!=0)
    {
      s = s+H[x];
    }
  }
```

Scanned with CamScanner

```
Printf ("\n Sum of odd Index is %d", S);
for(x=0; x<num; x++)
  {
    if (x%2==0)
    {
      Prod = Prod * H[x];
    }
  }
  Printf ("\n Product of odd Index is %d", Prod);
  Printf ("\n Enter the value of m");
  Scanf ("%d", &m);
    for(x=0; x<num; x++)
    {
      if (H[x]%m==0)
      {
        Printf ("%d", H[x]);
      }
    }
  }
```

output:-

```
Enter the number of elements to be entered 5
Enter 5 Integers.
4 5 6 7 8
Sorted list in Ascending order:
4
5
6
7
8
```

Scanned with CamScanner

Alternate order is 4 6 8
Sum of odd Index is 12
Product of odd Index is 35
Enter the value of m.
2
4 6 8.

) write a recursive Program to Implement binary search.

```
# include <stdio.h>
# include <stdlib.h>
Void binary search (int array[], int n, int first, int last){
int mid.
if (first > last){ element.
 Print f(" Number is not found");
 }
 else{
    mid = (first+last)/2 ;
   if (array(mid) == n){
   Print f(" element is found at Index %d", mid);
    exit(0);
  }
  else if
     (array(mid) > n) {
   Binary search (array, first, mid-1);
  }
  else{
    Binary search(array, n, mid+1, last);
```

Scanned with CamScanner

```
    }
  }
}

Void main()
int array[50], begining, mid, end, z, n, number;
Printf("enter size of Array");
scanf("%d", &number);
Printf("enter the values in sorted sequence \n");
 for(z=0; z<number; z++)
 {
  scanf("%d", &array[z]);
 }
 beg=0;
 end=n-1;
 Printf("Enter a value to be search:");
 scanf("%d", &n);
  Binary search(array, n, beg, end);
 }
```

output:-

```
Enter the size of Array 5
Enter the values in sorted sequence
  9
  10
  11
  12
  13
Enter a value to search: 10
Element found at Index 1
```

Scanned with CamScanner