Searching And Sorting

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.

```
#include <stdio.h>
{
int i, low, high, mid, n, key, arr[100], tmp, j, one, two, sum, product;
printf("Enter the number of elements in array");
scanf("%d", &n);
printf("Enter %d integers", n);
for (i=0; i<n; i++)
scanf("%d", &arr[i]);
for (i=0; i<n; i++)
{
for(j=0; j<i+1; j<n; j++)
{
 if (arr[i] < arr[j])
{
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
}
}
```

Scanned with CamScanner

```
printf("\n Elements of array is sorted in descending order:\n");
for (i=0 ; i<n ; i++)
{
printf ("%d ", arr[i]);
}
printf ("Enter valuee to find ");
Scanf ("%d", &key);
low = 0;
high = n-1;
mid = (low + high)/2;
while (low <= high)
{
if (arr[mid] > key)
low = mid + 1;
else if (arr[mid] == key) {
printf ("%d found at location %d", key, mid + 1);
break;
}
else
high = mid - 1;
mid = (low + high)/2;
}
if(low > high)
{
printf ("Notfound! %d isn't present in the list .n", key);
}
```

Scanned with CamScanner

b) Ask the user to enter any locations print the sum and product of values at those locations in the sorted array

```
printf ("\n");
printf ("enter two locations to find sum and product of the elements");
scanf("%d",&one);
scanf ("%d",&two);
sum = (arr[one] + arr[two]);
Product = (arr[one] * arr[two]);
printf (" The sum of elements = %d", sum);
printf("The product of elements = %d", product);
return 0;
}
```

Output :-

```
Enter number of elements in array 5
Enter 5 integers 9
7
5
4
2
Element of array is sorted in descending order :-
9 7 5 4 2  Enter value to find 5
5 found at location 3
Enter two locations to find sum and product of the elements 2
4
The sum of elements = 7
The product of elements = 10
```

Scanned with CamScanner

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

```
#include <stdio.h>
#include <conio.h>
#define MAX_SIZE 5
void merge_sort (int, int);
void merge_array (int,int,int,int);
int arr_sort[MAX_SIZE];
int main() {
int i, k, Poo = 1;
Printf (" Simple Merge sort Example Functions and Array \n");
Printf("\n Enter %d Elements for sorting \n", MAX_SIZE);
for (i=0 ; i < MAX_SIZE ; i++)
Scanf ( "%d", &arr_sort[j]);
Printf ["\n your Data:");
for (i=0 ;i< MAX_SIZE; i++) {
Printf ("\t %d", arr_sort [i]);
}
merge_sort (0, MAX_SIZE-1);
Print f["\n\n Sorted Data :");
for (i=0 ; i< MAX_SIZE ; i++){
Printf (" \t %d", arr_sort [i]);
}
Print f["Find the product of Kth elements from first and last where k\n");
```

Scanned with CamScanner

```
scanf ("%d", &k);
pro-arr_sort[k]*arr_sort[MAX_SIZE-k-1];
printf ("Product = %d", Pro);
getch( );
}
void merge_sort(int i, int j){
int m;
if (i<j) {
m = (i+j)/2;
merge-sort(i,m);
merge-sort (m+1,j);
// merging two arrays
merge-sort array(i, m, m+1, j);
}
}
void merge_array(int a, int b, int c, int d) {
int t[50];
int i=a ;j=c ;k=0;
while (i<=b && j<=d) {
if (arr_sort[i] < arr_sort[j])
t [k++] = arr-sort[i++];
else
  t [k++] = arr_sort [j++];
}
// collect remaining elements
while (i<=b)
  t [k++] = arr_sort[i++];
```

Scanned with CamScanner

```
while (j <= d)
    t[k++] = arr_sort[j++];
for (i = a, j = a; i <= d; i++, j++)
arr_sort[i] = t[j];
}
```

Output :-

```
Simple Merge Sort Example - Functions and Array
Enter 5 Elements for sorting
9
7
4
6
2
Your Data: 9 7 6 4 2
Sorted Data: 2 4 6 7 9
Find the product of the kth elements from first and last where k
2
Product = 36
```

3. Discuss Insertion and Selection sort with examples.

* Definition of 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 whole array is sorted in same order. The primary concept behind insertion sort is to insert each item into

Scanned with CamScanner

its appropriate place in the final of list. The insertion sort method saves an effective amount of memory.

Working of the Insertion Sort:-

→ It uses two sets of arrays where one stores the sorted data and other on unsorted data.

→ The sorting algorithm works until there are elements in the unsorted set.

→ Let's assume there are n number elements in the array. Initially, the element with index 0 (LB = 0) exists in the sorted set remaining elements are in the unsorted partition of the list.

→ The first element of the unsorted portion has very array index (if LB = 0)

→ After each iteration, it chooses the first element of the unsorted partition and inserts it into the proper place in the sorted set.

Advantages of Insertion Sort:-

→ Easily implemented and very efficient when used with small sets of data.

→ The additional memory space requirement of insertion sort is less (ie $o(1)$).

→ It is considered to be live sorting technique as the list can be sorted as the new elements are received.

→ It is faster than other sorting algorithms.

Scanned with CamScanner

Example:

25	15	30	9	99	20	26
15	25	30	9	99	20	26
15	25	30	9	99	20	26
9	15	25	30	99	20	26
9	15	25	30	99	20	26
9	15	20	25	30	99	26
9	15	20	25	26	30	99

☐ unsorted list ☐ sorted list

Definition of 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. The process of searching the minimum key and placing it in the proper position is continued until the all elements are placed at right position.

Working of the 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 then, the 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

Scanned with CamScanner

Interchange the ARR[POS] with ARR[1].

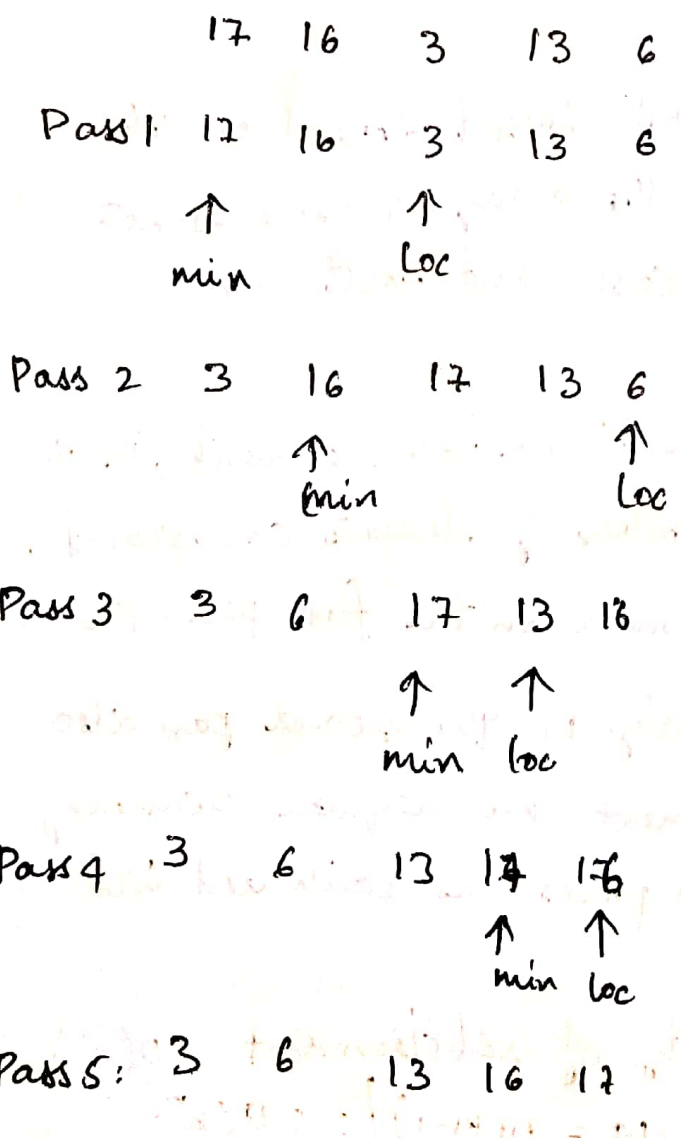
→ In the pass N-1, the same process is performed to sort the N number of elements.

Advantages of Selection Sort:-

→ The main advantage of selection sort is that it performs well on a small list

→ Further more, because it is an in-place sorting algorithm; no additional temporary storage is required beyond what is needed to hold the original list.

Example:-



Scanned with CamScanner

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 be compared with each element in the sorted set. So, in the worst case, running time of insertion sort is quadratic ie $O(n^2)$. In average case also it has to make the minimum $(k-1)/2$ Comparisons. Hence, the average case also has quadratic running time $O(n^2)$.

Complexity of Selection Sort:-

As the working of selection sort doesn't 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 ~~case~~ sort.

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

Thus, running time complexity of selection sort is $O(n^2)$

$= (n-1) + (n-2) + \cdots + 2 + 1 + 0 = n(n-1)/2 = O(n^2)$

Scanned with CamScanner

4

Sort the array using bubble sort where elements are taken from the user and display the elements 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 the user

```
#include <stdio.h>
#include <conio.h>
int main()
{
int arr[50];;, i, n, temp, sum=0, product = 1;
printf("Enter total number of elements to store:");
scanf("%d", &n);
printf("Enter %d elements:", n);
for (i=0; i<n; i++)
Scanf("%d", &arr[i]);
printf("\n Sorting array using bubble sort technique \n");
for (i=0; i<(n-1); i++)
{
 for (j=0; j<(n-1-1); j++)
{
if (arr[j] > arr[j+1])
{
temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
```

Scanned with CamScanner

```
}
}
}
printf ("All array elements sorted successfully \n");
printf("Array elements in ascending order: \n\n");
for (i= 0; i<n; i++) {
printf ("%d \n", arr[i]);
}
printf ("array elements in alternate order \n");
for (i=0 ; i<n ; i = i + 2){
printf ("%d \n", arr[i]);
}
for (i = 1; i<=n; i=i+2) {
sum = sum + arr[i];
}
printf ("The sum of odd position element are %d \n", sum);
for(i=0 ; i<=n ; i = i+2 )
{
product *= arr[i];
}
print f ("The product of even position elements are = %d \n",
                                                product);
get ch( );
return ( );
}
```

Scanned with CamScanner

Output:

```
Enter total number of elements to store : 5
Enter 5 elements : 8
 6
 4
 3
 2
Sorting array using bubble sort technique
All array elements sorted successfully:
Array elements in ascending order:
 2
 3
 4
 6
 8
array elements in alternate order
 2
 4
 8
The sum of odd position element are =9
The product of even position elements are =64
```

5. Write a recursive program to implement binary search

```
#include <stdio.h>
# include <stdlib.h>
void Binary search (int arr[.], int num, int first, in t last)
 {
 int mid;
if (first > last ){
printf ('Number is not found');
}
```

Scanned with CamScanner

```
else {
/* Calculate mid element */
mid = (first + last) / 2;
//If mid is equal to number we are searching
if (arr[mid] == num) {
printf ("Element is found at index %d", mid);
exit (0);
}
else {
 Binary Search (arr, num, mid+1, last);
 }
}
}
void main () {
  int arr[100], beg, mid, end, i, n, num;
printf ("Enter the size of an array");
scanf ("%d", &n);
printf ("Enter the values in sorted sequence \n");
for (i=0; i<n; i++)
{
scanf ("%d", &arr[i]);
}
 scanf ("%d", &arr[i]);
 }
 beg = 0;
 end = n-1;
```

Scanned with CamScanner

```
Printf ("Enter a value to be search:");
 Scanf ("%d", sum);
Binary Search (arr, num, beg, end);
}
```

Output:

```
Enter the size of an array 5
Enter the values in sorted sequence
 4
 5
 6
 7
 8
Enter a value to be search: 5
Element is found at index 1
```

Scanned with CamScanner