(1)

Bubble Sort

```
#include <stdio.h>
int main()
{
  int array [100], n, c, d, swap;
  printf ("Enter number of elements\n");
  scanf ("%d", &n);
  printf ("Enter %d integers\n", n);
  for (c=0; c<n; c++)
      scanf ("%d", &array [c]);
for (c =0; c <n -1; c++)
 {for (d=0; d<n -c-1; d++)
  {if (array [d] > array [d+1])
    {
      swap       = array [d];
      array [d]  = array [d+1];
      array [d+1] = swap;
    }
  }
}
```

```
printf("Sorted list in ascending order:\n");
for (c = 0; c < n; c++)
    printf("%d\n", array[c]);
    return 0;
}
```

Bubble Sort Algo:

Worst case performance O(n2)
Best case performance O(n), Average O(n2)

Linear Serch

```
#include <stdio.h>
int main()
{ int a[20], i, x, n;
  printf("How many elements?");
  scanf("%d", &n);
  printf("Enter array elements:\n");
    for (i = 0; i < n; ++i)
      scanf("%d", &a[i]);
```

```
printf("\nEnter element to search:");
scanf("%d", &x);
for(i=0; i<n; ++i)
  if(a[i] == x)
      break;
if(i<n)
    printf("Element found at index %d", i);
else printf("Element not found");
return 0;
}
```

Algo;

Worst case performance O(n)

Best case performance O(1)

Average case O(n)

(4)

Insertion Sort:

```
#include <stdio.h>
int main(){
    int i, j, count, temp, number[25];
printf("How many numbers u are going to enter?:");
 scanf("%d", &count);
printf("Enter %d element:", count);
for(i=0; i<count; i++)
   scanf("%d", &number[i]);
for(i=1; i<count; i++){
   temp = number[i];
   j = i-1;
   while((temp<number[j])&&(j>=0)){
      number[j+1] = temp;
   }
   printf("order of sorted elements:");
    for(i=0; i<count; i++)
       printf("%d", number[i]);
   return 0;
}
```

Algo:
Worst case Performance O(n2)
Best case performance O(n)
Average case O(n2)

Selection Sort:

```
#include <stdio.h>
int main(){
    int i, j, count, temp, number[25];
  printf("How many numbers u are going to enter:");
  scanf("%d", &count);
printf("Enter %d elements:", count);
 for(i=0; i<count; i++)
  scanf("%d", &number[i]);
for(j=0+i+j<count; j++){
  if(number[i] > number[j]){
      temp = number[i];
      number[i] = number[j]
      number[j] = temp;
} } }
```

```
printf ("Sorted elements:");
for (i=0; i<count; i++)
printf ("%d", number[i]);
return 0;
}
```

Algo.

Worst case performance $O(n^2)$

Best case performance $O(n^2)$

Average case performance $O(n^2)$

Binary Search:

```
#include<stdio.h>
int main()
{
int c, first, last, middle, n, search, array[100];
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
   scanf("%d", &search);
first = 0;
last = n - 1;
middle = (first + last) / 2;

while (first <= last) {
   if (array[middle] < search)
      first = middle + 1;
else if (array[middle] == search) {
   printf("%d found at location %d.\n", search,
                                        middle+1);
      break;
}
```

```
else
    last = middle - 1;
middle = (first + last)/2;
}
if (first > last)
printf("Not found! %d isn't present in the list.\n", search);
return 0;
}
```

Algo:

Worst case performance O(log n)

Best case performance O(1)

Average case performance O(log n)

Quick Sort:

```
#include <stdio.h>
void quickSort (int number [25], int first, int last);
   int i, j, pivot, temp;
if (first < last) {
      pivot = first;
       i = first;
       j = last;
while (number [i] <= number [pivot] && i < last)
       i++;
    while (number [j] > number [pivot])
       j--;
    if (i < j) {
                temp = number [i];
               number [i] = number [j];
               number [j] = temp;
        }
}
    temp = number [pivot];
    number [pivot] = number [j];
    number [j] = temp;
    quickSort (number, first, j-1);
```

```
quicksort (number, j+1, last);
}
}
int main ()
{
    int i, count, number [25];
printf ("How many elements are u going to enter?:");
scanf ("%d", &count);
printf ("Enter %d elements :", count);
for (i=0; i<count; i++)
    scanf ("%d", &number[i]);
quicksort (number, 0, count-1);
printf ("order of sorted elements:");
for (i=0; i<count; i++)
    printf ("%d", number[i]);
return 0;
}
```

worst case performance $O(n^2)$
Best case performance $O(n)$
Average case performance $O(n \log n)$