Assignment 8 Solution:

Exercise 1 (10 points)

Describe an algorithm that takes as input a list of n integers and produces as output the largest sum obtained by adding an integer in the list to the one following it.

```
int maxsum(int A[])
{
int n=A.length ;
int max=0;
int sum;
if (n<2)
        return;

   for(int i=0;i<n-1;i++)
   {

         sum=a[i+1]+a[i];

         if(sum>max)
         {
            max=sum;
         }

   }
   return max;
}
```

Exercise 2 (10 points)

Describe an algorithm that takes as input a list of n integers and finds the number of integers greater than 7 in the list.

```
int greaterThanSeven(int A[])
{
   Int n=A.length;
   int count=0;

   for(int i=0;i<n;i++)
   {
         if(a[i]>7)
         {
            count++;
         }
   }
   return count;
}
```

Exercise 3 (10 points)

Devise an algorithm to compute a^n, where a is a real number and n is an integer. [Hint:

First give a procedure for computing an when n is nonnegative by successive

multiplication by a, starting with 1. Then extend this procedure, and use the fact that a – n = 1/a n to compute a n when n is negative.]

```
double power(double a, int n)
{
 if(n==0)
 {
    return 1;
 }
   else if(n>=1)
   {
      return a*power(a,n-1);
   }
   else if (n<=-1)
   {
      return (1/a)*power(a,n+1);
```

}}

Exercise 4 (10 points)

Specify the steps of an algorithm that locates an element in a list of increasing integers by successively splitting the list into 3 sublists of equal (or as close to equal as possible) size, and restricting the search to the appropriate piece.

```
int search(int A[],int x)

{n=A.length;

i =1;

j =n;
int ind1, ind2=0;
While i<j
        ind1= ((i+j)/3);
        ind2= (2*(i+j))/3;

       if A[ind1]==x
              return ind1;

       if A[ind2]==x
              return ind2;

       else if A[ind1]> x
              j=ind1 -1;

       else if A[ind1]<x
       {
              if A[ind2]<x

                     i=ind2+1


              if A[ind2]> x
              {
                     i=ind1+1
                     j=ind2-1
              }
```

```
}}
return -1
```

Exercise 5 (10 points)

Devise an algorithm that finds all terms of a finite sequence of integers that are greater than the product of all previous terms of the sequence.

```
Void greaterThanProduct(int A[])
{
   int n=A.length;
   int product=1;

   for(int i=0;i<n;i++)
   {
      if(A[i]>prod)
      {
         Print A[i];
      }
      Product*=A[i];
   }
}
```

Exercise 6 (10 points)

List all the steps used to search for 4 and for 10 in the sequence 1, 3, 4, 5, 6, 8, 9, 11

using a) linear search and b) binary search.

a- Linear search:

Search for 4:

- The algorithm compares 4 with the first element 1. They are not equal, it continues searching.

  1, 3, 4, 5, 6, 8, 9, 11

- Then, it compares 4 and the second element 3, and there is still no match, so it continues.

  1, **3**, 4, 5, 6, 8, 9, 11

- Finally, it compares 4 to the third element 4. They are equal, then return the index of the third element which is 2.

  1, 3, **4**, 5, 6, 8, 9, 11

Search for 10:

- It compares 10 with every element in the list, one at a time, but it finds no match reaching the end of the list, so it returns -1 for there is no 10 in the list.

b- Binary Search:

Search for 4:

- 4 is compared with the element in the middle at index ⌊(size of list)/2 ⌋= ⌊8/2⌋=4 which is element 6. 6 is greater than 4, so, since the list is sorted in ascending order, 4 will be, most likely found, in the first half of the list whose first index is zero and size is half of the size of the list for its elements are all less than 6.

  1, 3, 4, 5, **6**, 8, 9, 11

- 4 is compared with the element at index ⌊(size of list)/4⌋=2 which is 4, there is a match so it returns index 2.

  1, 3, **4**, 5, 6, 8, 9, 11

Search for 10:

- 10 is similarly compared with the element at the middle which is 6. Since 10 is greater than 6, If it exists, it will be located in the second half of the list which starts on the middle +1 =size/2+1=5 (starts at index 5 at element 8) and ends at the end of the initial list.

  1, 3, 4, 5, **6**, 8, 9, 11

- Then, 10 is compared to the element in the middle of this part that is the element at index ⌊ (3*size of list)/4⌋=⌊ 3*8/4⌋=6 which is 9. Since 10 is greater than 9, it's most likely to be found in the part starting at index 6+1 which is 7 and ends at the end of the list.

  1, 3, 4, 5, 6, 8, **9**, 11

- This part of the list consists of only element 11, so the middle element is at index 7 which is 11 itself that is greater than 10. Then if 10 is found, it must be found in the part preceding to 11 which starts at index 7 and ends at 7-1=6 which means that there are no elements remaining in this part. Therefore, the list does not contain any 10. Return -1.

  1, 3, 4, 5, 6, 8, 9, **11**

Exercise 7 (10 points)

Sort q, f, t, l, a, d showing the lists obtained at each step using a) bubble sort and b)insertion sort.

a- Using Bubble Sort:

1- f,q,t,l,a,d

2- f,q,t,l,a,d

3- f,q,l,t,a,d

4- f,q,l,a,t,d

5- f,q,l,a,d,t

6- f,q,l,a,d,t

7- f,l,q,a,d,t

8- f,l,a,q,d,t

9- f,l,a,d,q,t

10- f,l,a,d,q,t

11- f,a,l,d,q,t

12- f,a,d,l,q,t

13- f,a,d,l,q,t

14- f,a,d,l,q,t

15- a,f,d,l,q,t

16- a,d,f,l,q,t

b- Using Insertion Sort:

1- f,q,t,l,a,d

2- f,q,t,l,a,d

3- f,l,q,t,a,d

4- a,f,l,q,t,d

5- a,d,f,l,q,t

Exercise 8 (10 points)

Describe an algorithm based on the binary search for determining the correct position in which to insert a new element in an already sorted list.

```
int binaryIndex(int x,int A[])
{
   int i=0;
   int j=A.length;
   int m=0;
   while(i<j)
   {
      m=(i+j)/2;
     if(x>a[m])
     {
        i=m+1;
     }
      else
      {
         j=m;
      }
   }

   return i;

  }
```

Exercise 9 (10 points)

Show that if there were a coin worth 12 cents, the greedy algorithm described in class using quarters, 12-cent coins, dimes, nickels, and pennies would not always produce change using the fewest coins possible.

Assume we had 20 cents for example:

The optimal solution would be:

2 dimes= 2*10=20;

On the other hand, by the greedy algorithm:

1 12-cent coin, 1 nickel, 3 pennies**.**

As the greedy algorithm will first choose the 12-cent coin for it is the one with the highest value less than or equal to 20. The amount remaining will be 20-12=8 cents, then, the algorithm will choose a nickel for it is the coin with the highest value less than or equal to 8. The amount remaining is 3 cents which is only changed by 3 pennies. Therefore, the greedy algorithm could not produce a change with the fewest coins possible.

Proved!

Exercise 10 (10 points)

Show that a greedy algorithm that schedules a set of talks in a lecture hall by selecting at each step the talk that overlaps the fewest with other talks does not always produce an optimal schedule.

Consider:

First talk:8:00-9:30

Second talk:9:00-10:00

Third talk:9:45-11:00

Fourth talk:8:30-9:30

By the greedy algorithm that schedules the set of talks by selecting at each time the talk that overlaps the fewest with the others, the solution will be the third talk only.

However, we can choose any of the 2 talks(first and fourth) with the third talk, so we get more talks in the hall than those given by the greedy algorithm.

Therefore, the greedy algorithm for this scheduling problem does not produce an optimal solution.