CS 601- Advance Algorithm
Homework 2

Name: Apurva Siddappa Raj
NetID: rv6455

<u>Solution 1:</u>

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
int i, j, n;
cout<<" Enter the number of elements in the list- ";
cin >> n;
cout<<"Enter elements: ";
int arr[n], count=0;
for(i = 0;i < n;i++)
{
cin >> arr[i];
}
for(i=0;i<n;i++)
{
for(j = i+1;j < n;j++)
{
if(arr[i] == arr[j])
{
   count++;
   break;
}
}
}
cout<<"Total number of Duplicates found- "<<count;
return 0;
}
```

Analyzing algorithm in worst case:

As there are two for loops, the outer for loop executes n times. Every time the outer loop executes, the inner loop also executes n times. Thus, the complexity is O(n*n). So, in the big O notation, it is O(n^2).

<u>Solution 2:</u>

```
#include<iostream>
#include<bits/stdc++.h>
```

```
using namespace std;
int main()
{
cout<<"Enter the number of elements in the list";
int n;
cin>>n;
cout<<"Enter elements";
int* list=new int[n];
for(int i=0;i<n;i++)
cin>>list[i];

// Sort Blackbox used to sort the input

sort(list,list+n);
int count=0;
int i=0, arr[i];
for(i=0;i<n;i++)
{
if(arr[i]==arr[i+1])
{
count++;
}
}
cout<<" Total number of Duplicates found -"<<count;
return 0;
}
```

Time Complexity:

Here I have sorted the list of numbers and scanned the array to find the duplicates.
All the elements will have a chance to run continuously. So, in a single scan we can find all the duplicates and also there is one loop which is running n times, so the worst case time complexity in a sorted array will be **n** (without considering the complexity of sorting). So, we can see that compared to time complexity of the question 1 of unsorted array(n^2) the time complexity in sorted array is improved to (n).

Solution 3a:

Arranging in ascending order of growth rate:

√2n < 2n+100 < nlogn < (n^2)(logn) < n^2.5 < n^10 < 2^n < 10^n < 100^n < n! < n^n

Solution 3b:

Proving f(n)=O(g(n))

For f2(n) and f9(n):

$f2(n) = \sqrt{2n} = O(n^{0.5})$; and $f9(n) = 2n+100 = O(n)$;
$\sqrt{2n} <= 2n+100$
So, $f2(n) < f9(n)$
for all values of $n>=1$ and $c = 1$
$(2n)^{0.5} <= 2n +100$ // f2(n) has less degree of 0.5 compared to f9(n)
Therefore, $f2(n) = O(f9(n))$

<u>For f9(n) and f7(n):</u>

$f9(n) = 2n+100 = O(n)$ and $f7(n) = nlogn = O(nlogn)$
$2n+100 <= nlogn$
For n>= 251 we can see that 2n+100 is bigO(nlogn)
So $f9(n) = O(f7(n))$ for c=1 and n>= 251

<u>For f7(n) and f8(n):</u>

$f7(n) = nlogn$ and $f8(n) = n^2(logn)$
$nlogn <= (n^2)(logn)$
We can see for all values of n>= 2 the above equation satisfies.
So $f7(n) = O(f8(n))$

<u>For f8(n) and f1(n):</u>

$f8(n) = (n^2)(logn)$ and $f1(n) = n^{2.5}$
$f8(n) < f1(n)$
taking log on both side: $2logn+ log(logn) < 2logn + 0.5logn$ (n>=1)
$log(logn) < 0.5logn$ { constant*log(n) is always greater than log(logn)}
So $f8(n) = O(f1(n))$

<u>For f1(n) and f3(n):</u>

$F1(n)=n^{2.5}$ and $f3(n)= n^{10}$
2.5 degree is less than 10
n>=1, $f1(n) = O(f3(n))$

<u>For f3(n) and f6(n):</u>
$F3(n)=n^{10}$ and $f6(n)=2^n$
Taking log on both sides: $10logn <= nlog2$. ignoring constants 10 and log2 we get
$logn <= n$ (n>=1)
so, n^{10} is BigO(2^n).

<u>For f6(n) and f4(n):</u>

$F6(n)=2^n$ and $f4(n)= 10^n$
Take log on both sides we get : $nlog2 <= nlog10$
log 10 is greater than log 2
2^n is BigO(10^n)
So $f6(n) = O(f4(n))$ c=1,for n>= 1

For f4(n) and f5(n):

f4(n) = 10^n and f5(n) = 100^n
Taking log on both sides: nlog10 <= nlog100
log 100 is greater than log 10
10^n is BigO(100^n)
f4(n) = O(f5(n)) for c=1, n>= 1

For f5(n) and f11(n):

F5=100^n and f(11)=n!
log(n!)= nlogn – n
log(100^n) = nlog100 ,
divide equations (nlogn – n)/(nlog100) = (logn/log100) – (1/log100)
1/log100 can be ignored
Hence for n is greater than 100
(100^n) =O(n!)

For f11(n) and f10(n):

F11(n)=n! and f10(n)= n^n
Let n>= 2
2! <= 2^2
2<=4
So f11(n) = O(f10(n)) for n>= 1 & c=1