Chapter 6

Searching

6.1 What is searching?

- Searching is the process used to find the location of a target among a list of objects.

6.2 Sequential Searching

- Sequential search is the simplest form of search.
- A.k.a. linear search.
- The sequential search procedure can be stated as starting at the beginning of the file, going through the records sequentially by examining the key values until the record with the given search key is found or the end of the file is reached and then stop. In the former case search is said to be successful (target found) and the in the later unsuccessful (target not found).
- It is used whenever the list is not ordered.
- Generally, you will use this technique only for small lists or lists that are not searched often.

- This search algorithm requires 4 parameters:
  a) The list we are searching
  b) An index to the last element in the list
  c) The target and
  d) The found element's index location

- One example of the algorithm might be:

```
cin>>target; found = 0;
for(i=0; i<n; i++)
{
```

```
        if(array[i] == target)
              found = 1;

}
```

- With this algorithm the running time is always dependent on the size of the array **n**. it does not matter where the target is, because the algorithm goes through all the element ALWAYS.

- Consider a different version

```
int i = 0, result = -1;
bool found = 0;
cin>>target;
 while( (i<n) && (found ==0)
)
{     if(array[i] ==
target)
      {
            result =i;
      found = 1;
      }
      i++;


}
```

- This algorithm returns the target's index location when successful and returns -1 when unsuccessful.
- Now the running time of this algorithm depends not only on **n** but also on the position of the target – if it is there.

- To try and analyse the algorithm we might look at:
  - ✓ The best case (i.e. when the target is first)
  - ✓ The average case (i.e. when the target is in the middle)
  - ✓ The worst case (i.e. when the target is at the end)

- Generally we will always compare algorithms on a worst case basis because this is really what we want to know – how long will it take?

- For instance in the running of a nuclear power plant in the event of some failure you need to know the longest time it might take to shut the plant down.

6.3 Binary Search

- As we have seen previously the sequential search is executed in linear time – that means the time is linearly dependent on the number of entries, n, in the table.

- As much more efficient search algorithm is the binary search. This algorithm relies on the elements in the table being in increasing order. In other word, a binary search can be performed only on ordered lists.

- The binary search uses the "divide and conquer" technique to search the list.

- When performing binary search, the array or list being searched will be divided in half after each cycle. It roughly divides the array in half after each attempt to find the target.

- The binary search starts by testing the data in the element at the middle of the array with the target key. If the target key is equal then the search ends. Otherwise, either halves (first half or second half) of the array are to be search in the same manner. In other words, we eliminate half the list from further consideration.

- We repeated this process until we find the target or determine that it is not in the list.
- To find the middle of the list, we need three variables: -
  a) one to identify the beginning of the beginning of the list => first
  b) one to identify the middle of the list => mid
  c) one to identify the end of the list. => last

- We can calculate mid point of the list as follow:

  mid = (first + last) / 2

- The program is as follows:

```
#include <iostream.h> void
main()
{
```

```
        int table [200]; int
        target,first,last,mid; int
        i; int found;

        //Setting up the array
        for(i=0; i<200; i++)
        {
              table[i] = 2*i;
        } first =
        0; last =
        199; found
        = 0;

        //Input the target
        cout<<“Input the target”;
        cin>>target;

        //The binary search while( (found==0)
        && (first <= last) )
        {     mid = (first + last) / 2;
              if( target == table[mid])
                    found = 1;
              else
              {
                    if( target<table[mid] )
                          last = mid – 1;
                    else
                          first = mid + 1;
              }
        }
        if(found)
              cout<<“\nPosition =”<<mid;
        else
              cout<<“\nTarget not found”;
}
```

- The above program assumes 200 elements in the table and their values are assigns from 0 to 398.

6.4 Analysis of the Binary Search

- Let us assume that the number of entries in the table is a power of 2. then the maximum number of iterations is going to be $\log_2 n$ (log to base 2 of n)

- For example if n=64, the number of iteration is 6 as the size of the table reduces as 32, 16, 8, 4, 2, 1

- In the above algorithm we do two comparisons per iteration, thus we have $2 \log_2 n$ comparisons.

- In terms of asymptotic notation we would say that for the binary search the growth is $O(\log_2 n)$. This is far superior to the sequential search which is O(n).