**EXPERIMENT 2**

**TITLE**: DIVIDE AND CONQUER I

1. **Implement Binary search and linear search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.**

**Code: (BINARY SEARCH)**

#include<stdio.h>

#include<time.h>

int binarysearch(int A[],int n,int el);

int binarysearch(int A[],int n,int el)

{

int l,u,mid,flag=0;

l=0;

u=n-1;

while(l<u)

{

mid=(l+u)/2;

if(el>A[mid])

l=mid+1;

else if (el<A[mid])

u=mid-1;

else

{

flag=1;

break;

}

}

}

int main()

{

clock\_t start,end;

double t; int i,n,flag,el;

printf("Program for Binary Search!\n");

printf("PLease note that binary searc assumes that the array is already sorted.\n Hence please enter the array values accordingly else this program might fail.\n");

printf("Enter the array size you want: ");

scanf("%d",&n);

int A[n];

for(i=0;i<n;i++)

{

printf("Enter the %d element of the array: ",i+1);

scanf("%d",&A[i]);

}

printf("Enter the element you want to search: ");

scanf("%d",&el);

printf("Performing the binary search\n");

start=clock();

flag=binarysearch(A,n,el);

end=clock();

if(flag==0)

printf("Element does not exist in the array\n");

else

printf("Element exist in the array\n");

t=(double)end-start/CLOCKS\_PER\_SEC;

printf("Time taken by the process in seconds is : %f\n",t);

return 0;

}

**Code: (LINEAR SEARCH)**

#include<stdio.h>

#include<time.h>

int linearsearch(int A[],int el,int n)

{

int i,flag=0;

for(i=0;i<n;i++)

{

if(A[i]==el)

{

flag=1;

break;

}

}

return flag;

}

int main()

{

int n,i,el,flag;

clock\_t start,end;

double time;

printf("Enter the number of elements in the array: ");

scanf("%d",&n);

int A[n];

for(i=0;i<n;i++)

{

printf("Enter the %d element of the array: ",i+1);

scanf("%d",&A[i]);

}

printf("Enter the element you want to search in the array: ");

scanf("%d",&el);

start=clock();

flag=linearsearch(A,el,n);

end=clock();

time = (double)start-end/CLOCKS\_PER\_SEC;

if(flag==1)

printf("The element exists in the array");

else

printf("The element does not exist in the array\n");

printf("Time taken for linear search is :%f\n",time);

return 0;

}

1. **Search a elements using the Binary search method and determine the time required to search the element. Repeat the experiment for different values of n, to search for the element in the list and plot a graph of the time taken versus n.**

**Code:**

#include<stdio.h>

#include<time.h>

int binarysearch(int A[],int n,int el);

int binarysearch(int A[],int n,int el)

{

int l,u,mid,flag=0;

l=0;

u=n-1;

while(l<u)

{

mid=(l+u)/2;

if(el>A[mid])

l=mid+1;

else if (el<A[mid])

u=mid-1;

else

{

flag=1;

break;

}

}

}

int main()

{

clock\_t start,end;

double t; int i,n,flag,el;

printf("Program for Binary Search!\n");

printf("PLease note that binary searc assumes that the array is already sorted.\n Hence please enter the array values accordingly else this program might fail.\n");

printf("Enter the array size you want: ");

scanf("%d",&n);

int A[n];

for(i=0;i<n;i++)

{

printf("Enter the %d element of the array: ",i+1);

scanf("%d",&A[i]);

}

printf("Enter the element you want to search: ");

scanf("%d",&el);

printf("Performing the binary search\n");

start=clock();

flag=binarysearch(A,n,el);

end=clock();

if(flag==0)

printf("Element does not exist in the array\n");

else

printf("Element exist in the array\n");

t=(double)end-start/CLOCKS\_PER\_SEC;

printf("Time taken by the process in seconds is : %f\n",t);

return 0;

}

1. **Sort a given set of elements using the Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

**Code:**

#include<iostream>

#include<time.h>

using namespace std;

//l is lowest index and r is the max index (left and right)

void merge(int A[],int l,int m ,int r)

{

//this function is to merge the partitioned arrays

//creating temp arrays

int n1 = m-l+1;

int n2 = r-m;

int A1[n1],A2[n2],i,j,k;

//First subarray begins from l to m

//second subarray begins from m+1 to r

//Filling the array

for(i=0;i<n1;i++)

{

A1[i] = A[l+i];

}

for(j=0;j<n2;j++)

{

A2[j] = A[m+1+j];

}

//Refilling the old array with correct order

i=0; //counter for A1

j=0;//counter for A2

k=l;//counter for A

while((i<n1) && (j<n2))

{

if (A1[i]<=A2[j])

{

A[k]=A1[i];

i++;

}

else

{

A[k]=A2[j];

j++;

}

k++;

}

//If any elements remain fill them because above while loop ends when either of the subarryas is filled

while(i<n1)

{

A[k]=A1[i];

i++;

k++;

}

while(j<n2)

{

A[k]=A2[j];

k++;

j++;

}

}

///////////////////////////////////////////////////////////////////

//Main merge sort function

void mergesort(int A[],int l,int r)

{

if(l<r)

{

int m=(l+r)/2;

//Sorting to first and second halves

mergesort(A,l,m);

mergesort(A,m+1,r);

//merging them

merge(A,l,m,r);

}

}

int main()

{

int n,l,r,i;

clock\_t start,end;

float t;

cout<<"Enter the array size you want: ";

cin>>n;

int A[n];

for(i=0;i<n;i++)

{

cout<<"Enter "<<i+1<<" element of the array: ";

cin>>A[i];

}

cout<<"Lets do merge sorting"<<endl;

start=clock();

mergesort(A,0,n-1);

end=clock();

for(i=0;i<n;i++)

{

cout<<i+1<<" element of the array is: "<<A[i]<<endl;

}

t=(float)end-start/CLOCKS\_PER\_SEC;

cout<<"Time taken to perform merge sort is: "<<t<<endl;

}