**EXPERIMENT III**

**TITLE:** DIVIDE AND CONQUER II

1. **Sort a given set of elements using the Quicksort method 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<stdio.h>

#include<time.h>

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

void swap(int \*x,int \*y)

{

int temp;

temp = \*x;

\*x = \*y;

\*y =temp;

}

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

int partition(int A[],int low,int high)

{

int i,j,pivot;

pivot = A[high];

i=low-1;

for(j=low;j<=high-1;j++)

{

if(A[j]<=pivot)

{

i++;

swap(&A[i],&A[j]);

}

}

swap(&A[i+1],&A[high]);

return (i+1);

}

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

void quicksort(int A[],int low,int high)

{

int pi;

if (low<high)

{

pi = partition(A,low,high);

quicksort(A,low,pi-1);

quicksort(A,pi+1,high);

}

}

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

int main()

{

clock\_t start,end;

double time;

int n,i;

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("Lets do the quick sort of given array\n");

start=clock();

quicksort(A,0,n-1);

end=clock();

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

printf("The sorted array is:\n");

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

{

printf("The %d element of the sorted array is: %d\n",i+1,A[i]);

}

printf("Time taken for the quick sort is: %f\n",time);

return 0;

}

1. **Sort a given set of elements using the insertion 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;

int main()

{

clock\_t s,e;

float t;

int n,i,j,key;

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

cin>>n;

int A[n];

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

{

cout<<"Enter "<<i+1<<" element of array"<<endl;

cin>>A[i];

}

cout<<"Lets do Insertion sort"<<endl;

s=clock();

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

{

key = A[i];

j=i-1;

while(j>=0 && A[j]>key)

{

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

j=j-1;

}

A[j+1]=key;

}

e=clock();

t=(float)e-s/CLOCKS\_PER\_SEC;

cout<<"Sorted Array is:"<<endl;

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

{

cout<<i+1<<" Element is :"<<A[i]<<endl;

}

cout<<"Time taken for processing is: "<<t<<endl;

}

1. **Implement Strassen’s matrix multiplication and compare the complexity with normal matrix multiplication**

**Code: (NAÏVE METHOD)**

#include<stdio.h>

#include<time.h>

int main()

{

int r1,c1,r2,c2,i,j,k;

clock\_t s,e;

float t;

printf("Enter the number of rows and columns of first matrix\n");

scanf("%d%d",&r1,&c1);

int A[r1][c1];

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

printf("Enter the %d row and %d column element of A matrix: ",i+1,j+1);

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

}

}

printf("The first matrix is\n");

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

printf("%d\t",A[i][j]);

}

printf("\n");

}

printf("Enter the number of rows and columns of second matrix\n");

scanf("%d%d",&r2,&c2);

int B[r2][c2];

for(i=0;i<r2;i++)

{

for(j=0;j<c2;j++)

{

printf("Enter the %d row and %d column element of B matrix: ",i+1,j+1);

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

}

}

printf("The second matrix is\n");

for(i=0;i<r2;i++)

{

for(j=0;j<c2;j++)

{

printf("%d\t",B[i][j]);

}

printf("\n");

}

printf("For the matrix multiplication process, we need number of columns of A matrix equal to the number of rows of B matrix\n");

s=clock();

if(c1==r2)

{

int C[r1][c2];

printf("Matrix multiplication process is possible for AxB\n");

for(i=0;i<r1;i++)

{

for(j=0;j<c2;j++)

{

C[i][j]=0;

for(k=0;k<c1;k++)

{

C[i][j]=C[i][j]+A[i][k]\*B[k][j];

}

}

}

printf("The muliplicative matrix is: \n");

for(i=0;i<r1;i++)

{

for(j=0;j<c2;j++)

{

printf("%d\t",C[i][j]);

}

printf("\n");

}

}

else

{

printf("Matrix multiplication operation is not possible\n");

}

e=clock();

t=(float)e-s/CLOCKS\_PER\_SEC;

printf("Time taken for processing is: %f\n",t);

return 0;

}

**Code: (STRASSEN’S MATRIX MULTIPLICATION)**

#include<stdio.h>

#include<time.h>

int main()

{

int A[2][2],B[2][2],C[2][2],i,j,p1,p2,p3,p4,p5,p6,p7;

clock\_t s,e;

float t;

printf("Enter the elements of matrix A\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

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

}

}

printf("Enter the elements of matrix B\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

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

}

}

s=clock();

p1=A[0][0]\*(B[0][1]-B[1][1]);

p2=(A[0][0]+A[0][1])\*B[1][1];

p3=(A[1][0]+A[1][1])\*B[0][0];

p4=A[1][1]\*(B[1][0]-B[0][0]);

p5=(A[0][0]+A[1][1])\*(B[0][0]+B[1][1]);

p6=(A[0][1]-A[1][1])\*(B[1][0]+B[1][1]);

p7=(A[0][0]-A[1][0])\*(B[0][0]+B[0][1]);

C[0][0]=p5+p4-p2+p6;

C[0][1]=p1+p2;

C[1][0]=p3+p4;

C[1][1]=p1+p5-p3-p7;

e=clock();

printf("Printing the array C\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

printf("%d\t",C[i][j]);

}

printf("\n");

}

t=(float)e-s/CLOCKS\_PER\_SEC;

printf("Time taken for processing is: %f\n",t);

return 0;

}