**Assignment#3**

**Algorithm: Multiply**

**Input:**

Two array representing digits of number1 and number2 as array\_num1[] and array\_num1[]

**Output:** Product of the numbers

1. Convert array\_num1[] and array\_num2[] to integer
2. if (num1 <= 10) or (num2 <= 10)

|  |
| --- |
| 1. return num1\*num2 2. Assign m🡨 maximum size from num1 and num2 3. Initialize mid🡨 m/2 4. a 🡨 num1 / 10^(mid) 5. b🡨 num1 % 10^(mid) 6. c 🡨 num2 / 10^(mid) 7. d 🡨 num2 % 10^(mid) 8. ac 🡨 Multiply(a,c) 9. bd 🡨 Multiply(b,d) 10. ad\_plus\_bc 🡨 Multiply(a+b,c+d) - ac - bd 11. prod = ac \* 10^(2\*mid) + (ad\_plus\_bc \* 10^mid) + bd 12. return prod |

**Multiply:**

def multiply(x,y):

if len(str(x)) <= 10 or len(str(y)) <= 10:

return x\*y

if len(str(x)) == 1 or len(str(y)) == 1:

return x\*y

else:

n = max(len(str(x)),len(str(y)))

mid = n / 2

a = x / 10\*\*(mid)

b = x % 10\*\*(mid)

c = y / 10\*\*(mid)

d = y % 10\*\*(mid)

ac = multiply(a,c)

bd = multiply(b,d)

ad\_plus\_bc = multiply(a+b,c+d) - ac - bd

prod = ac \* 10\*\*(2\*mid) + (ad\_plus\_bc \* 10\*\*mid) + bd

return prod

len\_x=input();

len\_y=input();

x=[]

y=[]

for i in range (0,len\_x):

x.append(input())

for i in range (0,len\_y):

y.append(input())

a,b=0,0

for i in range(0,len\_x):

a=a+((10\*\*(len\_x-1-i))\*x[i]);

for i in range(0,len\_y):

print(multiply(a,b))

**Algorithm: Ternary Search**

**Input:** Sorted Array(arr) and number to be searched(x)

**Output:** Index of the number

1. Initialize l🡨0
2. Initialize r🡨(array\_size-1)
3. if (r >= l)
   1. Set mid1 = l + (r - l)/3
   2. Set mid2 = mid1 + (r - l)/3
   3. if (arr[mid1] == x)
      1. return mid1
   4. if (arr[mid2] == x)
      1. return mid2
   5. if (arr[mid1] > x)
      1. return TernarySearch(arr, l, mid1-1, x)
   6. if (arr[mid2] < x)
      1. return ternarySearch(arr, mid2+1, r, x)
   7. return ternarySearch(arr, mid1+1, mid2-1, x)
4. else

4.1 return -1

1. end Algorithm

**Ternary Search:**

#include<stdio.h>

#include <stdlib.h>

#include <windows.h>

int function(long int arr[], long int l, long int r, long int x){

if (r >= l){

long int mid1 = l + (r - l)/3;

long int mid2 = mid1 + (r - l)/3;

if (arr[mid1] == x) return mid1;

if (arr[mid2] == x) return mid2;

if (arr[mid1] > x) return function(arr, l, mid1-1, x);

if (arr[mid2] < x) return function(arr, mid2+1, r, x);

return function(arr, mid1+1, mid2-1, x);

}

return -1;

}

long int main(void){

LARGE\_INTEGER frequency;

LARGE\_INTEGER start;

LARGE\_INTEGER end;

double interval;

long int n;

printf("Enter size of array:");

//long int n;

scanf("%lld",&n);

long int a[n];

printf("Enter Array elements:");

long long int i;

for(i=0;i<n;i++) a[i]=i;

printf("Enter no to be searched:");

long long int x;

scanf("%lld",&x);

long long int l=0;

long long int r=n-1;

long long int index;

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

index=function(a,l,r,x);

QueryPerformanceCounter(&end);

interval = ((double) (end.QuadPart - start.QuadPart) / frequency.QuadPart)\*1000;

printf("%f\n", interval);

if(index!=-1) printf("\n Index is: %lld",index);

else printf("\n Not Found.");

}

**Algorithm: TernarySearch(Modified Binary)**

**Input:** Sorted Array(arr) and number to be searched(x)

**Output:** Index of the number

1. Initialize l🡨0
2. Initialize r🡨(array\_size-1)
3. if (r >= l)
   1. Set mid1 = l + (r - l)/3
   2. if (arr[mid1] == x)
      1. return mid1
   3. if (arr[mid1] > x)
      1. return TernarySearch(arr, l, mid1-1, x)
   4. return ternarySearch(arr, mid1+1, mid2-1, x)
4. else

4.1 return -1

1. end Algorithm

**ModifiedBinarySearch:**

#include<stdio.h>

#include <stdlib.h>

#include <windows.h>

int function(int arr[], int l, int r, int x)

{

if (r >= l)

{

int mid1 = l + (r - l)/3;

if (arr[mid1] == x) return mid1;

if (arr[mid1] > x) return function(arr, l, mid1-1, x);

return function(arr, mid1+1, r, x);

}

return -1;

}

int main(void){

LARGE\_INTEGER frequency;

LARGE\_INTEGER start;

LARGE\_INTEGER end;

double interval;

int n;

printf("Enter size of array:");

//int n;

scanf("%d",&n);

int a[n];

printf("Enter Array elements:");

int i;

for(i=0;i<n;i++) scanf("%d",&a[i]);

printf("Enter no to be searched:");

int x;

scanf("%d",&x);

int l=0;

int r=n-1;

int index;

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

index=function(a,l,r,x);

QueryPerformanceCounter(&end);

interval = ((double) (end.QuadPart - start.QuadPart) / frequency.QuadPart)\*1000;

printf("%f\n", interval);

if(index!=-1) printf("\n Index is: %d",index);

else printf("\n Not Found.");

}

**Algorithm: MaxMin**

**Input:** Array

**Output:** output maximum and minimum

1. if (n == 1)
   1. return (A[1], A[1])
2. else if (n == 2)
   1. if( A[1] < A[2])
      1. return (A[1], A[2])
   2. else

2.1.1 return (A[2], A[1])

1. else
   1. (max\_left, min\_left) = maxmin(A[1...(n/2)])
   2. (max\_right, min\_right) = maxmin(A[(n/2 +1)...n])
   3. if (max\_left < max\_right)
      1. max = max\_right
   4. else
      1. max = max\_left
   5. if (min\_left < min\_right)
      1. max = min\_left
   6. else
      1. min = min\_right
   7. return (min, max)
2. end algorithm

**Max-Min:**

#include<stdio.h>

#include <stdlib.h>

#include <windows.h>

struct ret{ int min; int max; };

struct ret MaxMin(int arr[],int low,int high){

struct ret local\_result,left,right;

if(high-low==0){

local\_result.min=arr[low];

local\_result.max=arr[high];

return local\_result;

}

if(high-low==1){

if(arr[low]<arr[high]){

local\_result.min=arr[low];

local\_result.max=arr[high];

}

else{

local\_result.min=arr[high];

local\_result.max=arr[low];

}

return local\_result;

}

int mid;

mid=(low+high)/2;

left=MaxMin(arr,low,mid);

right=MaxMin(arr,mid+1,high);

if(left.min<right.min) local\_result.min=left.min;

else local\_result.min=right.min;

if(left.max>right.max) local\_result.max=left.max;

else local\_result.max=right.max;

return local\_result;

}

int main(void){

int N;

printf("Enter size of array\n");

scanf("%d",&N);

printf("Enter array:");

int i,a[N];

for(i=0;i<N;i++){

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

}

struct ret result;

result=MaxMin(a,0,N-1);

printf("\nMax is: %d\n",result.min);

printf("\nMin is: %d\n",result.max);

}

**Algorithm: MinMax**

**Input:** Array, low\_index, high\_index

**Output:** output max\_difference, result.min, result.max.

1. High🡨high-1
2. if(lo == hi)
3. return p[hi]-p[lo]
4. mid 🡨 (lo+hi)/2
5. (left.min,left.max) 🡨 MinMax(p,lo,mid);
6. (right.min,right.max) 🡨 MinMax (p,mid+1,hi)
7. set local\_min 🡨 p[mid]
8. for i = mid to lo do

8.1 if (p[i] < local\_min)

* + 1. local\_min = p[i]

1. set local\_max 🡨 p[mid+1]
2. for i=mid+1 to hi do

10.1 if(p[i] > local\_max)

* + 1. local\_max = p[i];

1. cross.min 🡨 local\_min
2. cross.max 🡨 local\_max
3. leftdiff 🡨 left.max - left.min
4. rightdiff 🡨 right.max - right.min
5. crossdiff 🡨 cross.max - cross.min
6. if(leftdiff > rightdiff and leftdiff > crossdiff)
   1. return (left.min,left.max)
7. else if(rightdiff > crossdiff)
   1. return (right.min,max)
8. else
   1. return (cross.min,cross.max)
9. EndAlgorithm

**MaximumJump:**

#include <stdio.h>

struct ret{ int min; int max; };

struct ret crossminmax(int p[],int lo,int mid,int hi){

int i,local\_min,local\_max;

struct ret local\_result;

local\_min = p[mid];

for(i=mid;i>=lo;i--){

if(p[i] < local\_min) local\_min = p[i]; }

local\_max = p[mid+1];

for(i=mid+1;i<=hi;i++){

if(p[i] > local\_max) local\_max = p[i]; }

local\_result.min = local\_min;

local\_result.max = local\_max;

return local\_result;

}

struct ret minmax(int p[],int lo,int hi){

int mid,leftdiff,rightdiff,crossdiff;

struct ret left,right,cross,local\_result;

if(lo == hi){

local\_result.min = p[lo]; local\_result.max = p[hi]; return local\_result; }

mid = (lo+hi)/2; left = minmax(p,lo,mid); right = minmax(p,mid+1,hi);

cross = crossminmax(p,lo,mid,hi);

leftdiff = left.max - left.min;

rightdiff = right.max - right.min;

crossdiff = cross.max - cross.min;

if(leftdiff > rightdiff && leftdiff > crossdiff) return left;

else if(rightdiff > crossdiff) return right;

else return cross;

}

int main(void){

int n,i;

printf("Enter size of array:");

scanf("%d",&n);

int arr[n];

printf("Enter array:\n");

for(i=0;i<n;i++) scanf("%d",&arr[i]);

struct ret result;

result = minmax(arr,0,n-1);

int temp=result.max - result.min;

printf("Max difference = %d\n",temp);

return 0;

}

**Algorithm: Fibo**

**Input:** Number(n)

**Output:** Nth Fibonacci number.

1. Set F[2][2] 🡨 {{1,1},{1,0}}
2. if (n == 0)
   1. return 0
3. power(F, n-1)
4. return F[0][0]
5. End Algorithm

Sub-Algorithm: power

Input: 2-D array(F), number(n).

Output: Void(Updated 2-D array F).

1. if( n == 0 || n == 1)
   1. return;
2. set M[2][2] 🡨 {{1,1},{1,0}}
3. power(F, n/2);
4. multiply(F, F);
5. if (n%2 != 0)

5.1 multiply(F, M);

1. End sub-algorithm

Sub-Algorithm: multiply

Input: 2-D array(F), 2-D array(M).

Output: Void(Updated 2-D array F).

1. x 🡨 F[0][0]\*M[0][0] + F[0][1]\*M[1][0]
2. y 🡨 F[0][0]\*M[0][1] + F[0][1]\*M[1][1]
3. z 🡨 F[1][0]\*M[0][0] + F[1][1]\*M[1][0]
4. w 🡨 F[1][0]\*M[0][1] + F[1][1]\*M[1][1]
5. F[0][0] 🡨 x
6. F[0][1] 🡨 y
7. F[1][0] 🡨 z
8. F[1][1] 🡨 w
9. End sub-algorithm

**Fibonacci:**

#include <stdio.h>

int fib(int n){

int F[2][2] = {{1,1},{1,0}};

if (n == 0) return 0;

power(F, n-1); return F[0][0];

}

void power(int F[2][2], int n){

if( n == 0 || n == 1) return;

int M[2][2] = {{1,1},{1,0}};

power(F, n/2);

multiply(F, F);

if (n%2 != 0) multiply(F, M);

}

void multiply(int F[2][2], int M[2][2]){

int x = F[0][0]\*M[0][0] + F[0][1]\*M[1][0];

int y = F[0][0]\*M[0][1] + F[0][1]\*M[1][1];

int z = F[1][0]\*M[0][0] + F[1][1]\*M[1][0];

int w = F[1][0]\*M[0][1] + F[1][1]\*M[1][1];

F[0][0] = x;

F[0][1] = y;

F[1][0] = z;

F[1][1] = w;

}

int main(void)

{

LARGE\_INTEGER frequency;

LARGE\_INTEGER start;

LARGE\_INTEGER end;

double interval;

printf("Enter n'th value:\n");

int n;

scanf("%d",&n);

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

int result=fib(n);

QueryPerformanceCounter(&end);

interval = ((double) (end.QuadPart - start.QuadPart) / frequency.QuadPart)\*1000;

printf("%f\n", interval);

printf("The %dth fibonacci number is: %d", n, result);

}

**Algorithm: Majority**

**Input:** Array, low\_index, high\_index

**Output:** output result.element, result.frequency.

1. if(high-low==0)
   1. return(arr[low],1)
2. if(low<high)
3. set mid🡨low+(high-low)/2
4. (left.elem,left.freq)🡨majority(arr,low,mid)
5. (right.elem,right.freq)🡨majority(arr,mid+1,high)
6. Set major🡨mid-low+1
7. Initialize count\_left🡨0,count\_right🡨0
8. if(right.freq!=0)

8.1 Set x🡨right.elem;

8.2 for(i=low to mid)

8.2.1 if(arr[i]==x)

8.2.1.1 Increment count\_left by 1

8.3 count\_left=count\_left+right.freq;

1. if(left.freq!=0)

9.1 Set x🡨left.elem;

9.2 for(i= mid+1 to high)

* + 1. if(arr[i]==x)

9.2.1.1 Increment count\_right by 1

9.3 count\_right🡨count\_right+left.freq;

1. if(count\_left>major)

10.1. return(right.elem ,count\_left)

1. else if(count\_right>major)

11.1. return(left.elem, count\_right)

1. else

12.1. return(-1,0)

1. End Algorithm

**Majority:**

#include<stdio.h>

struct ret{ int elem; int freq; };

struct ret majority(int arr[],int low,int high){

struct ret local\_result,left,right;

if(high-low==0){

local\_result.elem=arr[low]; local\_result.freq=1; }

if(low<high){

int mid=low+(high-low)/2; left=majority(arr,low,mid);

right=majority(arr,mid+1,high);

int major=mid-low+1; //condition for half+1

int count\_left=0,count\_right=0,x;

if(right.freq!=0){

x=right.elem;

for(i=low;i<=mid;++i){ if(arr[i]==x) count\_left++; }

count\_left=count\_left+right.freq; }

if(left.freq!=0){

x=left.elem;

for(i=mid+1;i<=high;++i){ if(arr[i]==x) count\_right++; }

count\_right+=left.freq; }

if(count\_left>major){

local\_result.freq=count\_left; local\_result.elem=right.elem; }

else if(count\_right>major){

local\_result.freq=count\_right; local\_result.elem=left.elem; }

else{

local\_result.elem=-1; local\_result.freq=0; }

} return local\_result;

}

int main(void){

int N;

printf("Enter size of array\n");

scanf("%d",&N);

printf("Enter array:");

int i,a[N];

for(i=0;i<N;i++) scanf("%d",&a[i]);

struct ret result;

result=majority(a,0,N-1);

if(result.freq!=0)

printf("The Majority element is= %d\twith frquency= %d\n",result.elem,result.freq);

else printf("No majority element");

}

**Algorithm: BinarySearch**

**Input:** Sorted Array(arr), low\_index, high\_index, number to be searched(x)

**Output:** Index of the number

1. if (r >= l)
   1. Set mid1 = l + (r - l)/2
   2. if (arr[mid1] == x)
      1. return mid1
   3. if (arr[mid1] > x)
      1. return BinarySearch(arr, l, mid1-1, x)
   4. return BinarySearch(arr, mid1+1, mid2-1, x)
2. else

4.1 return -1

1. end Algorithm

**Algorithm: IterationSearch**

**Input:** Sorted Array(arr), array\_size, number to be searched(x)

**Output:** Index of the number

1. for(i=0 to n-1) do

1.1. if(a[i]==x)

1.1.1 return i

1.2 else

1.2.1 return -1

2. End Algorithm

**Divide & Conquer:**

#include<stdio.h>

int binary(int arr[], int l, int r, int x){

if (r >= l){

int mid1 = l + (r - l)/2;

if (arr[mid1] == x) return mid1;

if (arr[mid1] > x) return binary(arr, l, mid1-1, x);

return binary(arr, mid1+1, r, x);

} return -1;

}

int iteration(int a[], int n, int x){

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

if(a[i]==x) return i;

}return -1;

}

int main(void){

int n;

printf("Enter size of array:"); scanf("%d",&n);

int a[n];

printf("Enter Array elements:");

for(i=0;i<n;i++) scanf("%d",&a[i]);

printf("Enter no to be searched:"); scanf("%d",&x);

int l=0, r=n-1,index\_iter,index\_bin;

index\_iter=iteration(a,n,x);

printf("-----Iteration Method---------");

if(index\_iter!=-1) printf("\n Index is: %d\n\n",index\_iter);

else printf("\n Not Found.\n\n");

index\_bin=binary(a,l,r,x);

printf("-----Divide & Conquer Method---------");

if(index\_bin!=-1) printf("\n Index is: %d\n",index\_bin);

else printf("\n Not Found.\n");

}