**Batch: B2 Roll No.: 16010121110**

**Experiment No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Binary search/Max-Min algorithm** |



**Objective:** To learn the divide and conquer strategy of solving the problems of different types



**CO to be achieved:**

|  |  |
| --- | --- |
| CO 2 | Describe various algorithm design strategies to solve different problems and analyse Complexity. |



**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Binary\_search\_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary\_search\_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**



**Pre Lab/ Prior Concepts:**

Data structures



**Historical Profile:**

Finding maximum and minimum or Binary search are few problems those are solved with the divide-and-conquer technique. This is one the simplest strategies which basically works on dividing the problem to the smallest possible level.

Binary Search is an extremely well-known instance of divide-and-conquer paradigm. Given an ordered array of n elements, the basic idea of binary search is that for a given element , "probe" the middle element of the array. Then continue in either the lower or upper segment of the array, depending on the outcome of the probe until the required (given) element is reached.



**New Concepts to be learned:**

Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.



**Algorithm IterativeBinarySearch**

int binary\_search(int A[ ], int key, int imin, int imax)

//The algorithm takes as parameters an array *A*[1.. *n*] , the search key and lower-higher index pair of the array.

// Output- The algorithm returns index of the search key in the given array, if it’s present.

{

// continue searching while [imin, imax] is not empty

**WHILE** (imax >= imin)

{

// calculate the midpoint for roughly equal partition

int imid = midpoint(imin, imax);

**IF**(A[imid] == key)

// key found at index imid

return imid;

// determine which subarray to search

**ELSE** **If** (A[imid] < key)

// change min index to search upper subarray

imin = imid + 1;

**ELSE**

// change max index to search lower subarray

imax = imid - 1;

}

// key was not found

**RETURN** KEY\_NOT\_FOUND;

}

**The Time and space complexity of Iterative Binary Search:**

O(log n)

**Algorithm Recursive Binary Search**

int binary\_search(int A[], int key, int imin, int imax)

//The algorithm takes as parameters an array *A*[1.. *n*] , the search key and lower-higher index pair of the array.

// Output- The algorithm returns index of the search key in the given array, if it’s present.

{

// test if array is empty

**IF** (imax < imin)

// set is empty, so return value showing not found

**RETURN** KEY\_NOT\_FOUND;

**ELSE**  {

// calculate midpoint to cut set in half

int imid = midpoint(imin, imax);

// three-way comparison

**IF** (A[imid] > key)

// key is in 🡨 lower subset

**RETURN** binary\_search(A, key, imin, imid-1);

**ELSE IF** (A[imid] < key)

// key is in 🡪 higher subset

**RETURN** binary\_search(A, key, imid+1, imax);

**ELSE**

// key has been found

**RETURN** imid;

}

}

**The Time and space complexity of Recursive Binary Search:**

O(log n)

**Algorithm StraightMaxMin:**

**VOID** StraightMaxMin (Type a[], int n, Type& max, Type& min)

// Set max to the maximum and min to the minimum of a[1:n].

{ max = min = a[1];

**FOR** (int i=2; i<=n; i++)

{

**IF** (a[i]>max) then max = a[i];

**IF** (a[i]<min) min = a[i];

}

}

**Algorithm: Recursive Max-Min**

**VOID** MaxMin(int i, int j, Type& max, Type& min)

// A[1:n] is a global array. Parameters i and j are integers, 1 <= i <= j <= n.

//The effect is to set max and min to the largest and smallest values in a[i:j], respectively.

{

**IF** (i == j)

max = min = a[i]; // Small(P)

**ELSE IF** (i == j-1) { // Another case of Small(P)

**IF** (a[i] < a[j])

max = a[j];

min = a[i];

**ELSE** { max = a[i]; min = a[j];

}

**ELSE** { Type max1, min1;

// If P is not small divide P into sub problems. Find where to split the set.

int mid=(i+j)/2;

// solve the sub problems.

MaxMin(i, mid, max, min);

MaxMin(mid+1, j, max1, min1);

// Combine the solutions.

**IF** (max < max1) max = max1;

**IF** (min > min1) min = min1;

}

}

**The space complexity of Max-Min:**

Log n

**Time complexity for Max-Min:**

Tn=2T(n/2)+2 O(n)

**Code -** [**https://github.com/Aatmaj-Zephyr/AOA/tree/main/Practical/Expt2**](https://github.com/Aatmaj-Zephyr/AOA/tree/main/Practical/Expt2)

**CONCLUSION:**

Thus we have implemented recursive and non-recursive versions of Max Min and Binary Search. We have analysed the time complexity for both the cases. We learnt how to find the time complexity for divide and conquer algorithms

**Calculation:**

**Max min**

**a=2 as both parts are subproblems**

**b=2 as two parts are divided**

**f(n)=2 = O(n^0) time required for merging is 2 as n comparisns are made for merging**

**By masters theorem, third case will be considered**

**Tn=O(n^log2 (2))=O(n)**

**Binary search**

**a=1 as only one of the two parts is a subproblems**

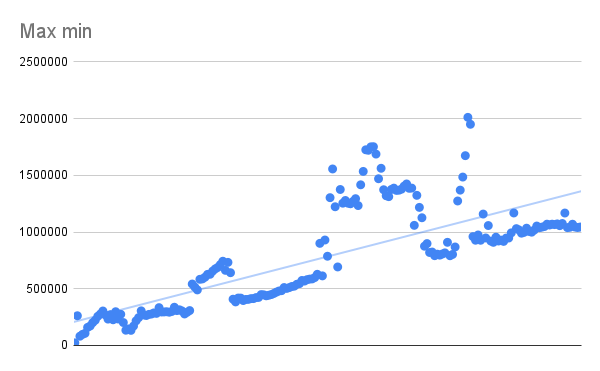
**b=2 as two parts are divided**

**f(n)=2 = O(n^0) time required for merging is 2 as n comparisns are made for merging**

**By masters theorem, second case will be considered**

**This is because n=0=log2 (1)**

**Hence O(log n)**

****

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Recursive Max,Min**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public class Main**

**{**

**public static void main(String[] args) {**

**System.out.println("Hello World");**

**int a[]={809,766,227,832,71,862,496,264,771,542,84,252,508,585,824,708,620,849,850,23,232,635,685,3,471,149,33,745,427,399,939,498,361,148,723,942,469,15,216,397,165,227,250,267,117,373,719,845,231,135,821,254,89,664,691,425,100,654,482,804,37,371,915,751,597,673,925,328,681,94,870,741,320,872,43,798,500,649,977,313,964,264,844,321,116,740,826,890,131,908,554,321,329,208,894,456,885,950,518,536,687,490,474,877,73,753,925,297,113,295,965,17,270,532,591,779,709,666,194,986,767,399,591,405,270,324,250,611,835,416,890,726,402,808,603,418,644,315,180,438,66,688,603,753,86,552,363,420,128,614,853,912,489,144,967,397,66,511,492,453,301,709,729,669,440,289,365,100,659,602,659,560,615,606,689,929,971,966,939,74,796,77,628,489,154,832,764,575,930,6,170,400,39,872,770,650,752,379,666,115};**

**for(int i=0;i<a.length-1;i++){**

**long initTime= System.nanoTime();**

**max(a,0,i);**

**min(a,0,i);**

**System.out.println( System.nanoTime()-initTime);**

**}**

**}**

**public static int max(int[]a,int start,int end){**

**if(start==end){**

**return a[start];**

**}**

**int mid=(int)((start+end)/2);**

**int b=max(a,start,mid);**

**int c=max(a,mid+1,end);**

**if(b>c){**

**return b;**

**}**

**else{**

**return c;**

**}**

**}**

**public static int min(int[]a,int start,int end){**

**if(start==end){**

**return a[start];**

**}**

**int mid=(int)((start+end)/2);**

**int b=min(a,start,mid);**

**int c=min(a,mid+1,end);**

**if(b<c){**

**return b;**

**}**

**else{**

**return c;**

**}**

**}**

**}**

**import java.util.\*;**

**public class Main {**

**public static void main(String[] args) {**

**int size=3000;**

**int[] arr = new int[size];**

**Random rd = new Random(); // creating Random object**

**for(int i=0;i<size;i++){**

**arr[i]=rd.nextInt(1000);**

**}**

**selectionSort(arr);**

**for(int i=0;i<=arr.length-1;i++){**

**long TS1 = System.nanoTime();**

**iterativeBinarySearch(arr, 0, i, arr[i],0);**

**long TS2 = System.nanoTime();**

**System.out.println(TS2 - TS1);**

**}**

**}**

**public static int[] selectionSort(int[] arr) {**

**for (int i = 0; i < arr.length - 1; i++) {**

**for (int j = i + 1; j < arr.length; j++) {**

**if (arr[i] < arr[j]) {**

**// swap**

**int temp = arr[j];**

**arr[j] = arr[i];**

**arr[i] = temp;**

**}**

**}**

**}**

**return arr;**

**}**

**public static int recursiveBinarySearch(int[] arr, int start, int end, int key,int times) {**

**int mid = (int) ((start + end) / 2);**

**if (arr[mid] == key) {**

**// System.out.println(times+1);**

**return mid;**

**} else if (start > end) {**

**return -1;**

**}**

**if (key > arr[mid]) {**

**return recursiveBinarySearch(arr, start, mid - 1, key,times+1);**

**}**

**if (key <= arr[mid]) {**

**return recursiveBinarySearch(arr, mid + 1, end, key,times+1);**

**}**

**return 0;// never**

**}**

**public static int iterativeBinarySearch(int[] arr, int start, int end, int key,int times) {**

**while (true) {**

**int mid = (int) ((start + end) / 2);**

**if (arr[mid] == key) {**

**// System.out.println(times+1);**

**return mid;**

**}**

**if (arr[mid] > key) {**

**start = mid + 1;**

**}**

**if (arr[mid] < key) {**

**end = mid - 1;**

**}**

**if (start > end) {**

**return -1;**

**}**

**times++;**

**}**

**}**

**}**