|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **24 / 01 / 2023** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Prof. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 2**

**Title: Binary search and Finding Minimum and Maximum.**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To learn the divide and conquer strategy of solving the problems of different types. |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Describe various algorithm design strategies to solve different problems. |

|  |
| --- |
| **Apparatus / Software Tools Used:** |
| 1. VS Code 2. Microsoft Excel |

|  |
| --- |
| **Theory:** |
| **Historical Profile:**  Finding maximum and minimum or Binary search are a few problems that are solved with the divide-and-conquer technique. This is one the simplest strategies that works on dividing the problem to the smallest possible level.  Binary Search is an extremely well-known instance of the 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:**   1. Number of comparisons 2. Application of algorithmic design strategy to any problem     Classical problem-solving vs. Divide-and-Conquer problem-solving. |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| **Algorithm Iterative Binary Search:**  int binary\_search(int A[], int key, int imin, int imax)  {  // 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;  }  **Algorithm Recursive Binary Search:**  int binary\_search(int A[], int key, int imin, int imax)  {  // 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 the lower subset  return binary\_search(A, key, imin, imid - 1);  else if (A[imid] < key)  // key is in the higher subset  return binary\_search(A, key, imid + 1, imax);  else  // key has been found  return imid;  }  }  **Algorithm StraightMaxMin:**  void StraightMaxMin(Type a[], int n, Type& max, Type& min)  {  max = min = a[1];  for (int i = 2; i <= n; i++)  {  if (a[i] > max)  max = a[i];  if (a[i] < min)  min = a[i];  }  }  **Algorithm: Recursive Max-Min**  void MaxMin(int i, int j, Type& max, Type& min)  {  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 subproblems. Find where to split the set.  int mid = (i + j) / 2;  // solve the subproblems.  MaxMin(i, mid, max, min);  MaxMin(mid + 1, j, max1, min1);  // Combine the solutions.  if (max < max1)  max = max1;  if (min > min1)  min = min1;  }  } |

|  |
| --- |
| **Codes / Outputs:** |
| 1. **Binary Search:**   #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int binary\_search(int arr[], int key, int imin, int imax)  {      while (imax >= imin)      {          int imid = (imin + imax) / 2;          if (arr[imid] == key)          {              printf("element %d found at index %d.\n", key, imid);              return imid;          }          else if (arr[imid] < key)              imin = imid + 1;          else              imax = imid - 1;      }      printf("element not found", key);      return -1;  }  void insertion\_sort(int arr[], int n)  {      int i, j, temp;      for(i = 1; i < n; i++)      {          temp = arr[i];          j = i-1;          while((temp < arr[j]) && (j>=0))          {              arr[j+1] = arr[j];              j--;          }      arr[j+1] = temp;      }  }  int main()  {      clock\_t start, end;      double cpu\_time\_used;      int n, key;      printf("\nA - 2 / 16014022050");      printf("\nenter number of elements: ");      scanf("%d", &n);      printf("\n");        int arr[n];      srand(time(NULL));      for (int i = 0; i < n; i++)      {          arr[i] = rand() % (n + 1);      }      printf("unsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      insertion\_sort(arr, n);      printf("\nsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      printf("\nenter element to search in array: ");      scanf("%d", &key);      printf("\n");        start = clock();      binary\_search(arr, key, 0, n - 1);      end = clock();      cpu\_time\_used = ((double)(end - start)) / (CLOCKS\_PER\_SEC / 1000000);      printf("\ntime taken: %.2f microseconds\n", cpu\_time\_used);      return 0;  }     1. **Recursive Binary Search:**   #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int binary\_search\_recur(int arr[], int key, int imin, int imax)  {      if (imax >= imin)      {          int imid = (imin + imax) / 2;          if (arr[imid] == key)          {              printf("element %d found at index %d\n", key, imid);              return imid;          }          else if (arr[imid] < key)              return binary\_search\_recur(arr, key, imid + 1, imax);          else              return binary\_search\_recur(arr, key, imin, imid - 1);      }      printf("element not found\n", key);      return -1;  }  void insertion\_sort(int arr[], int n)  {      int i, j, temp;      for(i = 1; i < n; i++)      {          temp = arr[i];          j = i-1;          while((temp < arr[j]) && (j>=0))          {              arr[j+1] = arr[j];              j--;          }      arr[j+1] = temp;      }  }  int main()  {      clock\_t start, end;      double cpu\_time\_used;      int n, key;      printf("\nA - 2 / 16014022050");      printf("\nenter number of elements: ");      scanf("%d", &n);      printf("\n");        int arr[n];      srand(time(NULL));      for (int i = 0; i < n; i++)      {          arr[i] = rand() % (n + 1);      }      printf("unsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      insertion\_sort(arr, n);      printf("\nsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      printf("\nenter element to search in array: ");      scanf("%d", &key);      printf("\n");        start = clock();      binary\_search\_recur(arr, key, 0, n - 1);      end = clock();      cpu\_time\_used = ((double)(end - start)) / (CLOCKS\_PER\_SEC / 1000000);      printf("\ntime taken: %.2f microseconds\n", cpu\_time\_used);      return 0;  }     1. **Straight Max-Min:**   #include <stdio.h>  #include <stdlib.h>  #include <time.h>  void maxMin\_stright(int arr[], int n, int \*max, int \*min)  {      \*max = arr[0];      \*min = arr[0];      for (int i = 1; i < n; i++)      {          if (arr[i] > \*max)              \*max = arr[i];            if (arr[i] < \*min)              \*min = arr[i];      }  }  int main()  {      clock\_t start, end;      double cpu\_time\_used;      int n;      printf("\nenter number of elements: ");      scanf("%d", &n);        int arr[n];      srand(time(NULL));      for (int i = 0; i < n; i++)      {          arr[i] = rand() % (n + 1);      }      printf("\nunsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      int max, min;      start = clock();      maxMin\_stright(arr, n, &max, &min);      end = clock();      printf("\nsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      printf("\nmax: %d\n", max);      printf("min: %d\n", min);      cpu\_time\_used = ((double)(end - start)) / (CLOCKS\_PER\_SEC / 1000000);      printf("\ntime taken: %.2f microseconds\n", cpu\_time\_used);      return 0;  }     1. **Recursive Max-Min:**   #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int binary\_search\_recur(int arr[], int key, int imin, int imax)  {      if (imax >= imin)      {          int imid = (imin + imax) / 2;          if (arr[imid] == key)          {              printf("element %d found at index %d\n", key, imid);              return imid;          }          else if (arr[imid] < key)              return binary\_search\_recur(arr, key, imid + 1, imax);          else              return binary\_search\_recur(arr, key, imin, imid - 1);      }      printf("element not found\n", key);      return -1;  }  void insertion\_sort(int arr[], int n)  {      int i, j, temp;      for(i = 1; i < n; i++)      {          temp = arr[i];          j = i-1;          while((temp < arr[j]) && (j>=0))          {              arr[j+1] = arr[j];              j--;          }      arr[j+1] = temp;      }  }  int main()  {      clock\_t start, end;      double cpu\_time\_used;      int n, key;      printf("\nA - 2 / 16014022050");      printf("\nenter number of elements: ");      scanf("%d", &n);      printf("\n");        int arr[n];      srand(time(NULL));      for (int i = 0; i < n; i++)      {          arr[i] = rand() % (n + 1);      }      printf("unsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      insertion\_sort(arr, n);      printf("\nsorted array: \n");      for (int i = 0; i < n; i++)      {          printf("%d ", arr[i]);      }      printf("\n");      printf("\nenter element to search in array: ");      scanf("%d", &key);      printf("\n");        start = clock();      binary\_search\_recur(arr, key, 0, n - 1);      end = clock();      cpu\_time\_used = ((double)(end - start)) / (CLOCKS\_PER\_SEC / 1000000);      printf("\ntime taken: %.2f microseconds\n", cpu\_time\_used);      return 0;  } |

|  |
| --- |
| **Observation Table:** |
| 1. **Graphs for varying input sizes for Binary Search:**        1. **Graphs for varying input sizes for Recursive Binary Search:**        1. **Graphs for varying input sizes for Straight Max-Min:**        1. **Graphs for varying input sizes for Recursive Max-Min:** |

|  |
| --- |
| **Calculations:** |
| 1. **The space complexity of Recursive Binary Search:** 2. **The Time complexity of Binary Search:** 3. **The space complexity of Max-Min:** 4. **Time complexity for Max-Min:** |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Solve the problem theoretically which was implemented during the practical.** |

|  |
| --- |
| **Conclusion:** |
| In conclusion, this experiment helped us understand the binary search and Max-Min algorithms and how it efficiently organizes/searches data, iteratively as well as recursively. After careful analysis, we determined its time and space complexities, offering valuable insights into its practical effectiveness across different situations. |

**Signature of faculty in-charge with Date:**