**Experiment 10**

(PART B : TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| Program: BTI CE | Sem: VII |
| Roll No. C013 | Name: Ashmit Jain |
| Division: B | Batch : B1 |
| Date of Experiment: | Date of Submission: 28/10/2024 |
| Grade : |  |

**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Binary Search Code:**

#include <iostream>

using namespace std;

bool BinarySearch(int arr[], int low, int high, int key)

{

if(low<=high)

{

int mid = (low + high) / 2;

if(arr[mid] == key)

{

return true;

}

else if(arr[mid] > key)

{

return BinarySearch(arr, low, mid-1, key);

}

else

{

return BinarySearch(arr, mid+1, high, key);

}

}

return false;

}

int main()

{

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter the elements: ";

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

{

cin >> arr[i];

}

int key;

cout << "Enter the element to be searched: ";

cin >> key;

int low = 0, high = n-1;

if ( BinarySearch(arr, low, high, key) == true )

{

cout << "Element found" << endl;

}

else

{

cout << "Element not found" << endl;

}

return 0;

}

**Insertion Sort Code:**

#include <iostream>

using namespace std;

void insertionSort(int arr[], int n)

{

for (int i = 1; i < n; i++)

{

int key = arr[i];

int j = i - 1;

// Move elements of arr[0...i-1] that are greater than key

// one position ahead to make space for the key

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

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int main()

{

int arr[] = {5, 4, 3, 1, 2};

int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

// Print the sorted array directly in main

cout << "Sorted array: ";

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

{

cout << arr[i] << " ";

}

cout << endl;

return 0;

}

Merge Sort

#include <iostream>

using namespace std;

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1; // Size of the left subarray

int n2 = right - mid; // Size of the right subarray

int leftArr[n1], rightArr[n2];

// Copy data to temporary arrays

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

leftArr[i] = arr[left + i];

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

rightArr[j] = arr[mid + 1 + j];

// Merge the temporary arrays back into arr[left...right]

int i = 0, j = 0, k = left;

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

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

i++;

} else {

arr[k] = rightArr[j];

j++;

}

k++;

}

// Copy any remaining elements of leftArr, if any

while (i < n1) {

arr[k] = leftArr[i];

i++;

k++;

}

// Copy any remaining elements of rightArr, if any

while (j < n2) {

arr[k] = rightArr[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

// Sort first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr) / sizeof(arr[0]);

mergeSort(arr, 0, n - 1);

// Print the sorted array

cout << "Sorted array: ";

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

cout << arr[i] << " ";

cout << endl;

return 0;

}

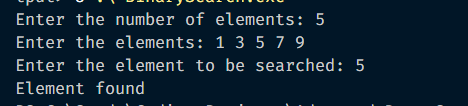
**B.2 Input and Output:**

***(Paste your commented program input and output in following format, If there is error then paste***

***the specific error in the output part. In case of error with due permission of the faculty extension***

***can be given to submit the error free code with output in due course of time.)***

**Binary Search**

****

**Insertion Sort**

****

****

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

 **Binary Search**:

* Efficient for large, sorted datasets with O(log⁡N)O(\log N)O(logN) complexity, but requires the array to be pre-sorted.
* Reduces search space quickly, ideal for static data where minimal changes occur.

 **Insertion Sort**:

* Simple, intuitive, and works well for small or nearly sorted datasets with O(n2)O(n^2)O(n2) complexity.
* Stable and adaptive, making it efficient in cases with partially sorted data.

 **Merge Sort**:

* Consistent O(nlog⁡n)O(n \log n)O(nlogn) performance, suitable for large datasets.
* Uses additional memory for merging, but is stable and reliable, especially for linked lists or large arrays.

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

 **Binary Search** is optimal for static, sorted arrays due to its efficiency but requires pre-sorted data.

 **Insertion Sort** is practical for small datasets and partially sorted data due to its simplicity and adaptability.

 **Merge Sort** is a robust, stable choice for large datasets but uses more memory.

**B.5 Question of Curiosity**

**Show the working of Mergesort on following numbers**

**-8, 5, 1, 0, 5, 26, 47, 10,99, 23**

**Merge Sort Steps:**

1. **Initial Array**: [-8, 5, 1, 0, 5, 26, 47, 10, 99, 23]
2. **Divide the Array** (Repeatedly split the array in half until each sub-array has one element):
   * Split 1: [-8, 5, 1, 0, 5] and [26, 47, 10, 99, 23]
   * Split 2: [-8, 5] and [1, 0, 5] | [26, 47] and [10, 99, 23]
   * Split 3: [-8] and [5] | [1] and [0, 5] | [26] and [47] | [10] and [99, 23]
   * Split 4: [0] and [5] | [99] and [23]
3. **Merge Process** (Sort and merge sub-arrays back together):
   * **Merge Step 1**: Merge [-8] and [5] → [-8, 5]
   * **Merge Step 2**: Merge [0] and [5] → [0, 5]
   * **Merge Step 3**: Merge [1] and [0, 5] → [0, 1, 5]
   * **Merge Step 4**: Merge [-8, 5] and [0, 1, 5] → [-8, 0, 1, 5, 5]
   * **Merge Step 5**: Merge [26] and [47] → [26, 47]
   * **Merge Step 6**: Merge [99] and [23] → [23, 99]
   * **Merge Step 7**: Merge [10] and [23, 99] → [10, 23, 99]
   * **Merge Step 8**: Merge [26, 47] and [10, 23, 99] → [10, 23, 26, 47, 99]
   * **Merge Step 9**: Merge [-8, 0, 1, 5, 5] and [10, 23, 26, 47, 99] → [-8, 0, 1, 5, 5, 10, 23, 26, 47, 99]
4. **Final Sorted Array**: **[-8, 0, 1, 5, 5, 10, 23, 26, 47, 99]**

*(To be answered by student base****d on the practical performed and learning/observations)***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*