Name :Amruta Pede

Reg no:2020BIT070

Assignment 3

Write C/C++ code to implement concept of

1) Searching Algoritham (Any Three)

2) Sorting Algoritham (Any Three)

1)Searching Algorithm:

1)Linear Search:

```
// C++ code to linearly search x in arr[]. If x is present then return its location, otherwise
// return -1

#include <iostream>
using namespace std;
int search(int arr[], int N, int x)
{
	int i;
	for (i = 0; i < N; i++)
		if (arr[i] == x)
			return i;
	return -1;
}
// Driver's code
int main(void)
{
	int arr[] = { 2, 3, 4, 10, 40 };
	int x = 10;
```

```
    int N = sizeof(arr) / sizeof(arr[0]);
   // Function call
    int result = search(arr, N, x);
    (result == -1)
            ? cout << "Element is not present in array"
            : cout << "Element is present at index " << result;
    return 0;
}
```

Output:

main.cpp

```
#include <iostream>
using namespace std;
int search(int arr[], int N, int x)
{
    int i;
    for (i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
// Driver's code
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int N = sizeof(arr) / sizeof(arr[0]);
            // Function call
    int result = search(arr, N, x);
    (result == -1)
        ? cout << "Element is not present in array"
        : cout << "Element is present at index " << result;
    return 0;
}
```

Output

```
/tmp/y4vE9V8moV.o
Element is present at index 3
```

2)Binary Search:

```
// C++ program to implement iterative Binary Search
#include <bits/stdc++.h>
using namespace std;

// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
```

```
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
	while (l <= r) {
		int m = l + (r - l) / 2;

		// Check if x is present at mid
		if (arr[m] == x)
			return m;

		// If x greater, ignore left half
		if (arr[m] < x)
			l = m + 1;

		// If x is smaller, ignore right half
		else
			r = m - 1;
	}

	// if we reach here, then element was
	// not present
	return -1;
}

int main(void)
{
	int arr[] = { 5,7,9,18,27,45 };
	int x = 10;
	int n = sizeof(arr) / sizeof(arr[0]);
```

```
        int result = binarySearch(arr, 0, n - 1, x);
        (result == -1)
                ? cout << "Element is not present in array"
                : cout << "Element is present at index " << result;
        return 0;
}
```

Output :

main.cpp

```cpp
// Online C++ compiler to run C++ program online
// C++ program to implement iterative Binary Search
#include <bits/stdc++.h>
using namespace std;

// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }
```

Output

```
/tmp/y4vE9V8moV.o
Element is not present in array
```

3)Ternary Search :

```
 #include <iostream>
using namespace std;

// Function to perform Ternary Search
int ternarySearch(int l, int r, int key, int ar[])

{
        while (r >= l) {
```

```
// Find the mid1 and mid2
int mid1 = l + (r - l) / 3;
int mid2 = r - (r - l) / 3;

// Check if key is present at any mid
if (ar[mid1] == key) {
        return mid1;
}
if (ar[mid2] == key) {
        return mid2;
}

// Since key is not present at mid,
// check in which region it is present
// then repeat the Search operation
// in that region

if (key < ar[mid1]) {

        // The key lies in between l and mid1
        r = mid1 - 1;
}
else if (key > ar[mid2]) {

        // The key lies in between mid2 and r
        l = mid2 + 1;
}
else {
```

```
                // The key lies in between mid1 and mid2
                l = mid1 + 1;
                r = mid2 - 1;
            }
        }

        // Key not found
        return -1;
}

// Driver code
int main()
{
        int l, r, p, key;

        // Get the array
        // Sort the array if not sorted
        int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        // Starting index
        l = 0;

        // length of array
        r = 9;

        // Checking for 5

        // Key to be searched in the array
        key = 5;
```

```
    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    cout << "Index of "<<key<<" is " << p << endl;

    // Checking for 50

    // Key to be searched in the array
    key = 50;

    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    cout << "Index of "<<key<<" is " << p;
}
```

Output:

main.cpp

```
    r = 9;

    // Checking for 5

    // Key to be searched in the array
    key = 5;

    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    cout << "Index of "<<key<<" is " << p << endl;

    // Checking for 50

    // Key to be searched in the array
    key = 50;

    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    cout << "Index of "<<key<<" is " << p;
}
```

Output

```
/tmp/y4vE9V8moV.o
Index of 5 is 4
Index of 50 is -1
```

2)Sorting Algorithm:

1)Selection Sort:

```
// C++ program for implementation of
// selection sort
#include <bits/stdc++.h>
using namespace std;

//Swap function
void swap(int *xp, int *yp)
{
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}
```

```
void selectionSort(int arr[], int n)
{
	int i, j, min_idx;

	// One by one move boundary of
	// unsorted subarray
	for (i = 0; i < n-1; i++)
	{

		// Find the minimum element in
		// unsorted array
		min_idx = i;
		for (j = i+1; j < n; j++)
		if (arr[j] < arr[min_idx])
			min_idx = j;

		// Swap the found minimum element
		// with the first element
		if(min_idx!=i)
			swap(&arr[min_idx], &arr[i]);
	}
}

//Function to print an array
void printArray(int arr[], int size)
{
	int i;
	for (i=0; i < size; i++)
```

```
            cout << arr[i] << " ";
        cout << endl;
}

// Driver program to test above functions
int main()
{
        int arr[] = {64, 25, 12, 22, 11};
        int n = sizeof(arr)/sizeof(arr[0]);
        selectionSort(arr, n);
        cout << "Sorted array: \n";
        printArray(arr, n);
        return 0;
}
```

Output:

main.cpp

Run

```
            swap(&arr[min_idx], &arr[i]);
        }
}

//Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Output

```
/tmp/y4vE9V8moV.o
Sorted array:
11 12 22 25 64
```

2)Bubble sort:

```
// C++ program for implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
	int i, j;
	for (i = 0; i < n - 1; i++)

		// Last i elements are already
		// in place
		for (j = 0; j < n - i - 1; j++)
			if (arr[j] > arr[j + 1])
				swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
	int i;
	for (i = 0; i < size; i++)
		cout << arr[i] << " ";
	cout << endl;
}

// Driver code
```

```
int main()
{
        int arr[] = { 5, 1, 4, 2, 8};
        int N = sizeof(arr) / sizeof(arr[0]);
        bubbleSort(arr, N);
        cout << "Sorted array: \n";
        printArray(arr, N);
        return 0;
}
```

Output:

main.cpp

```
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8};
    int N = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);
    return 0;
}
```

Output

```
/tmp/y4vE9V8moV.o
Sorted array:
1 2 4 5 8
```

3)Quick Sort:

```
/* C++ implementation of QuickSort */
#include <bits/stdc++.h>
using namespace std;

// A utility function to swap two elements
void swap(int* a, int* b)
{
```

```
	int t = *a;
	*a = *b;
	*b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
	int pivot = arr[high]; // pivot
	int i
		= (low
		- 1); // Index of smaller element and indicates
				// the right position of pivot found so far

	for (int j = low; j <= high - 1; j++) {
		// If current element is smaller than the pivot
		if (arr[j] < pivot) {
			i++; // increment index of smaller element
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[high]);
	return (i + 1);
}
```

```
/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
	if (low < high) {
		/* pi is partitioning index, arr[p] is now
		at right place */
		int pi = partition(arr, low, high);

		// Separately sort elements before
		// partition and after partition
		quickSort(arr, low, pi - 1);
		quickSort(arr, pi + 1, high);
	}
}

/* Function to print an array */
void printArray(int arr[], int size)
{
	int i;
	for (i = 0; i < size; i++)
		cout << arr[i] << " ";
	cout << endl;
}

// Driver Code
int main()
```

```
{
        int arr[] = { 10, 7, 8, 9, 1, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, n - 1);
        cout << "Sorted array: \n";
        printArray(arr, n);
        return 0;
}
```

Output:

main.cpp

```
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver Code
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Run

Output

```
/tmp/y4vE9V8moV.o
Sorted array:
1 5 7 8 9 10
```