LQT Day 1 Sorting

🔍 Overview of Sorting Algorithms

1. Selection Sort

- **Idea**: Select the smallest element from the unsorted part and swap it with the first unsorted element.
- **Time Complexity**: O(n²)
- **Space Complexity**: O(1)
- **Stable**: No

🔧 **C++ Code:**

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

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int minIndex = i;
        for (int j = i+1; j < n; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        swap(arr[i], arr[minIndex]);
    }
}

int main() {
    int arr[] = {29, 10, 14, 37, 13};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    return 0;
}
```

For additional resource: https://www.geeksforgeeks.org/dsa/selection-sort-algorithm-2/

---

2. Insertion Sort

- **Idea**: Build sorted array one element at a time by inserting each new element into its proper position.

- **Time Complexity**: $O(n^2)$ (Best case O(n) if already sorted)
- **Space Complexity**: O(1)
- **Stable**: Yes

🔧 **C++ Code:**

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

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int current = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > current) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = current;
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr, n);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    return 0;
}
```

For additional Resources: https://www.geeksforgeeks.org/dsa/insertion-sort-algorithm/

---

3. Quick Sort

- **Idea**: Use divide-and-conquer strategy by picking a "pivot", partitioning the array, and recursively sorting subarrays.
- **Time Complexity**: O(n log n) average, $O(n^2)$ worst
- **Space Complexity**: O(log n)
- **Stable**: No

🔧 **C++ Code:**

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

```
int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++)
        if (arr[j] < pivot)
            swap(arr[++i], arr[j]);
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int p = partition(arr, low, high);
        quickSort(arr, low, p - 1);
        quickSort(arr, p + 1, high);
    }
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    return 0;
}
```

For additional resources:https://www.geeksforgeeks.org/dsa/quick-sort-algorithm/

---

4. Merge Sort

- **Idea**: Divide the array into halves, recursively sort both halves, and then merge the sorted halves.
- **Time Complexity**: O(n log n)
- **Space Complexity**: O(n)
- **Stable**: Yes

🔧 C++ Code:

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

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    mergeSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    return 0;
}
```

For additional Resources:

✅ Lab Quiz Test (LQT)

LQT Pattern

- **Practical:**
    - Q1: Implement Quick Sort on an array of size 10.
    - Q2: Implement Merge Sort and show the array after each merge step.
    - Q3: Write code to sort student scores using Selection Sort.

HACKER RANK

Use the link provided below to access Hacker rank challenge:

https://www.hackerrank.com/contests/sorting-1752852059/challenges

1. Sort an Array (LeetCode #912)

🧠 2. Sort Array by Parity (LeetCode #905)

Problem: Rearrange an integer array so that all even numbers come before all odd numbers.
Concepts: Two-pointer approach or stable partitioning. LeetCode

---

🧠 3. Sort Colors (LeetCode #75)

Problem: Sort an array containing only 0, 1, and 2 in one pass and in-place.
Concepts: Dutch National Flag algorithm — linear time, constant space. LeetCode

---

🧠 4. Sort Array by Increasing Frequency (LeetCode #1636)

Problem: Sort elements by increasing frequency; for ties, sort by decreasing value.
Concepts: Hash map + custom comparator + sorting. LeetCode

---

5. Sort the People (LeetCode #2418)

Problem: Sort names by their corresponding heights in descending order.
Concepts: Pairing and sorting with custom order. LeetCode