Heapsort in C++

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

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if(left < n && arr[left] > arr[largest])
        largest = left;

    if(right < n && arr[right] > arr[largest])
        largest = right;

    if(largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for(int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for(int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Step-by-Step Dry Run

✅ Step 1: Build Max Heap

Indices:

0: 12 1: 11 2: 13 3: 5 4: 6 5: 7

Start from i = 2 (last non-leaf node)

i	Heapify Subtree	Max-Heap after heapify
2	[13, 7]	No change
1	[11, 5, 6]	No change
0	[12, 11, 13, 5, 6, 7]	swap 12 with 13 → heapify(2) swaps 12 with 7 → Done

🔺 Max Heap Built:

[13, 11, 7, 5, 6, 12]

✅ Step 2: Extract Elements & Heapify

We now swap root with last element and reduce heap size (n--) after each step:

i	Swap arr[0] & arr[i]	Array after swap	Heapify to max heap
5	swap(13, 12)	[12, 11, 7, 5, 6, 13]	→ heapify → [11, 12, 7...] → [11, 6, 7, 5, 12, 13]
4	swap(11, 6)	[6, 5, 7, 11, 12, 13]	→ heapify → [7, 5, 6...]
3	swap(7, 5)	[5, 6, 7, 11, 12, 13]	→ heapify → [6, 5, ...]
2	swap(6, 5)	[5, 6, 7, 11, 12, 13]	→ heapify → [5, 6, ...] (already heap)
1	swap(5, 5)	Done	

✅ Final Output
Sorted array is
5 6 7 11 12 13

Sorted array is

5 6 7 11 12 13

Insertion Sort in C++

```
#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;
//        while(j>=0 && arr[j]>key){
//         arr[j+1]=arr[j];
//         j=j-1;
//        }
//        arr[j + 1] = key;
//     }
// }

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

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

Input:
arr[] = {12, 11, 13, 5, 6}

Step-by-Step Dry Run (Tabular Form)

i (loop index)	j (inner loop)	Comparison	Action	Array State
1	1	11 < 12	swap(11, 12)	[11, 12, 13, 5, 6]
2	2	13 < 12? ✗	no swap	[11, 12, 13, 5, 6]
3	3	5 < 13 → swap	[11, 12, 5, 13, 6]	
	2	5 < 12 → swap	[11, 5, 12, 13, 6]	
	1	5 < 11 → swap	[5, 11, 12, 13, 6]	
4	4	6 < 13 → swap	[5, 11, 12, 6, 13]	
	3	6 < 12 → swap	[5, 11, 6, 12, 13]	
	2	6 < 11 → swap	[5, 6, 11, 12, 13]	

✓ Final Output:
Sorted array:
5 6 11 12 13

5 6 11 12 13

MergeSort in C++

```
#include<bits/stdc++.h>
using namespace std;

void merge(int arr[], int l, int m, int r) {
    int n1=m-l+1;
    int n2=r-m;
    int left[n1];
    int right[n2];

    for(int i=0;i<n1;i++){
        left[i]=arr[l+i];
    }

    for(int j=0;j<n2;j++){
        right[j]=arr[m+1+j];
    }
    int i = 0, j = 0, k = l;

    while(i<n1 && j<n2){
        if(left[i]<=right[j]){
            arr[k]=left[i];
            i++;
        }else{
            arr[k]=right[j];
            j++;
        }
        k++;
    }

     while (i < n1) {
         arr[k]=left[i];
         i++;
         k++;
     }

    while(j<n2){
        arr[k]=right[j];
        j++;
        k++;
    }
}

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

int main() {
    /* Enter your code here. Read input from STDIN.
Print output to STDOUT */

    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
```

Example Input:
n = 6
arr = {38, 27, 43, 3, 9, 82}

Merge Sort Recursive Dry Run (Call Stack Overview)

Call Level	Function Call	Action	Array State
1	mergeSort(0, 5)	Split at 2	
2	mergeSort(0, 2)	Split at 1	
3	mergeSort(0, 1)	Split at 0	
4	mergeSort(0, 0)	Base case	[38]
4	mergeSort(1, 1)	Base case	[27]
3	merge(0, 0, 1)	Merge [38] & [27] → [27, 38]	[27, 38, 43, 3, 9, 82]
2	mergeSort(2, 2)	Base case	[43]
2	merge(0, 1, 2)	Merge [27, 38] & [43]	[27, 38, 43, 3, 9, 82]
1	mergeSort(3, 5)	Split at 4	
2	mergeSort(3, 4)	Split at 3	
3	mergeSort(3, 3)	Base case	[3]
3	mergeSort(4, 4)	Base case	[9]
2	merge(3, 3, 4)	Merge [3] & [9] → [3, 9]	[27, 38, 43, 3, 9, 82]
1	mergeSort(5, 5)	Base case	[82]
1	merge(3, 4, 5)	Merge [3, 9] & [82]	[27, 38, 43, 3, 9, 82]
0	merge(0, 2, 5)	Merge [27, 38, 43] & [3, 9, 82] →	

```
        cin >> arr[i];
    }

    mergeSort(arr,0,n-1);

    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

		[3, 9, 27, 38, 43, 82]	

✅ Final Output:
3 9 27 38 43 82

3 9 27 38 43 82

Quick Sort in C++

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

int medianOfThree(int arr[], int l, int h) {
    int mid = l + (h - l) / 2;
    if (arr[l] > arr[mid]) swap(arr[l], arr[mid]);
    if (arr[l] > arr[h]) swap(arr[l], arr[h]);
    if (arr[mid] > arr[h]) swap(arr[mid], arr[h]);
    return mid;
}

int partition(int arr[], int l, int h) {
    int medianIndex = medianOfThree(arr, l, h);
    swap(arr[l], arr[medianIndex]);  // Move median to start as pivot

    int pivot = arr[l];
    int left = l + 1;
    int right = h;

    while (left <= right) {
        while (left <= right && arr[left] < pivot) left++;
        while (left <= right && arr[right] > pivot) right--;

        if (left <= right) {
            swap(arr[left], arr[right]);
            left++;
            right--;
        }
    }

    swap(arr[l], arr[right]);  // Put pivot in correct place
    return right;
}

void rquicksort(int arr[], int l, int h) {
    if (l < h) {
        int pivot = partition(arr, l, h);
        rquicksort(arr, l, pivot - 1);
        rquicksort(arr, pivot + 1, h);
    }
}

int main() {
    int arr[] = {24, 97, 40, 67, 88, 85, 15};
```

Here's a **dry run** of your Quicksort code **in tabular form** for the input:

int arr[] = {24, 97, 40, 67, 88, 85, 15};

We'll trace:

- Recursive calls
- Chosen pivot (via median-of-three)
- Partitioning process
- Array state after each step

🔁 Step-by-Step Dry Run Table:

Step	Subarray (l to h)	Median-of-Three	Pivot	Final Pivot Index	Array After Partition
1	arr[0..6] = {24,97,40,67,88,85,15}	40 (mid=2)	40	2	{24,15,40,67,88,85,97}
2	arr[0..1] = {24,15}	15 (mid=0)	15	0	{15,24,40,...}
3	arr[1..1] = {24}	-	-	-	(Base case, already sorted)
4	arr[3..6] = {67,88,85,97}	85 (mid=4)	85	4	{...,67,85,88,97}
5	arr[3..3] = {67}	-	-	-	(Base case)
6	arr[5..6] = {88,97}	88 (mid=5)	88	5	{...,67,85,88,97} (already sorted)

int n = sizeof(arr) / sizeof(arr[0]);
 	
rquicksort(arr, 0, n - 1);
 	
cout << "Sorted array: ";
for (int i = 0; i < n; i++) {
cout << arr[i] << " ";
}
cout << endl;
 	
return 0;
}
	🧠 **Final Sorted Array:**
	
	{15, 24, 40, 67, 85, 88, 97}

15, 24, 40, 67, 85, 88, 97	

Selection in C++

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

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

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Input:

arr[] = {64, 25, 12, 22, 11}

📊 Selection Sort Dry Run Table

Pass	i	Initial minidx	Comparisons	New minidx	Swap (arr[i] ↔ arr[minidx])	Array after pass
1	0	0 (64)	25 < 64 → 12 < 25 → 22 < 12 → 11 < 12	4 (11)	64 ↔ 11	[11, 25, 12, 22, 64]
2	1	1 (25)	12 < 25 → 22 < 12	2 (12)	25 ↔ 12	[11, 12, 25, 22, 64]
3	2	2 (25)	22 < 25	3 (22)	25 ↔ 22	[11, 12, 22, 25, 64]
4	3	3 (25)	64 > 25	3	25 ↔ 25 (no change)	[11, 12, 22, 25, 64]

✅ Final Output:

Sorted array:
11 12 22 25 64

11 12 22 25 64