Lecture 14

M M Imran

Asymptotic Analysis

- The efficiency of an algorithm depends on the amount of time, storage and other resources required to execute the algorithm.

- An algorithm may not have the same performance for different types of inputs.

- With the increase in the input size, the performance will change.

Asymptotic Notations

- The efficiency is measured with the help of asymptotic notations.

- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm.

Asymptotic Notations

- Consider a sorting algorithm that sorts an array with arbitrary numbers. Let’s say we are considering Bubble sort

- When the input array is already sorted, the time taken by the algorithm is linear **(Best Case)**
- When the input array is in reverse condition, the algorithm takes the maximum time to sort the elements **(Worst Case)**
- When the input array is neither sorted nor in reverse order, then it takes average time **(Average Case)**

Different Asymptotic Notations

- Big-O notation
- Omega notation
- Theta notation

Big-O Notation (O-notation)

- Big-O notation represents the upper bound of the running time of an algorithm.

- Thus, it gives the **worst-case complexity** of an algorithm.

Omega Notation (Ω-notation)

- Omega notation represents the lower bound of the running time of an algorithm.

- Thus, it provides the **best-case** complexity of an algorithm.

Theta Notation (Θ-notation)

- It represents the upper and the lower bound of the running time of an algorithm

- Used for analyzing the **average-case** complexity of an algorithm.

Most of the time we just need to find out the **worst-case** complexity of an algorithm.

Time and Space complexity

- **Time Complexity** of an algorithm/code is not equal to the actual time required to execute a particular code, but the number of times a statement executes.

- **Space Complexity** of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space (extra space or temporary space used by an algorithm) and space used by input.

Time Complexity: O(1)

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

int main()
{
    cout << "Hello World";
    return 0;
}
```

Time Complexity: O(n)

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

int main()
{

    int i, n = 8;
    for (i = 1; i <= n; i++) {
        cout << "Hello World !!!\n";
    }
    return 0;
}
```

Time Complexity: $O(n^2)$

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

int main()
{
  int i, j, n = 8;
  for (i = 1; i <= n; i++) {
    for (j = 1 <= n; j++){
      cout << "Hello World !!!\n";
    }
  }
  return 0;
}
```

Time Complexity: $O(\log_2(n))$

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

int main()
{

    int i, n = 8;
    for (i = 1; i <= n; i=i*2) {
        cout << "Hello World !!!\n";
    }
    return 0;
}
```

What will be the Time Complexity?

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

int main()
{
  int i, j, k, n = 8;
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      for (k = 1; k <= n; k++ {
        cout << "Hello World !!!\n";
      }
    }
  }
  return 0;
}
```

Space Complexity: O(1)

```
int addSequence (int n){
    int sum = 0;
    for (int i = 0; i < n; i++){
        sum += pairSum(i, i+1);
    }
    return sum;
}

int pairSum(int x, int y){
    return x + y;
}
```

Space Complexity: O(n)

```
int add(int n) {
    if (n <= 0) {
        return 0;
    }
    return n + add (n-1);
}
```

```
1.  add(4)
2.      -> add(3)
3.          -> add(2)
4.              -> add(1)
5.                  -> add(0)
```

Sorting Algorithm

A sorting algorithm is used to arrange elements of an array/list in a specific order.

Different Sorting Algorithms

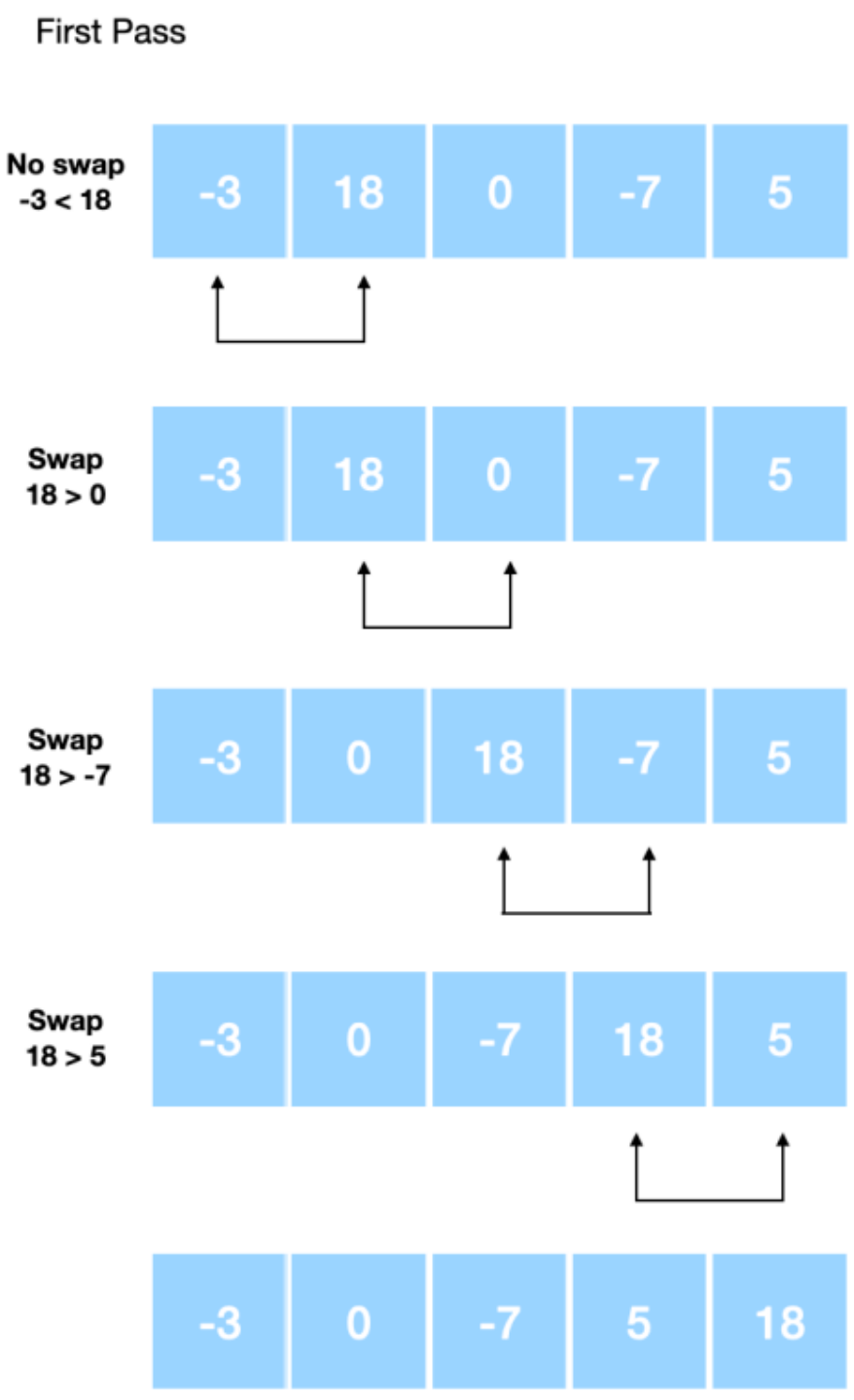
- Selection Sort
- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Recursive Insertion Sort
- Merge Sort
- Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort
- Bucket Sort
- Shell Sort

Bubble Sort

- Bubble Sort is the simplest of the sorting techniques.
- In the bubble sort technique, each of the elements in the list is compared to its adjacent element.
- Thus if there are n elements in list A, then A[0] is compared to A[1], A[1] is compared to A[2] and so on.

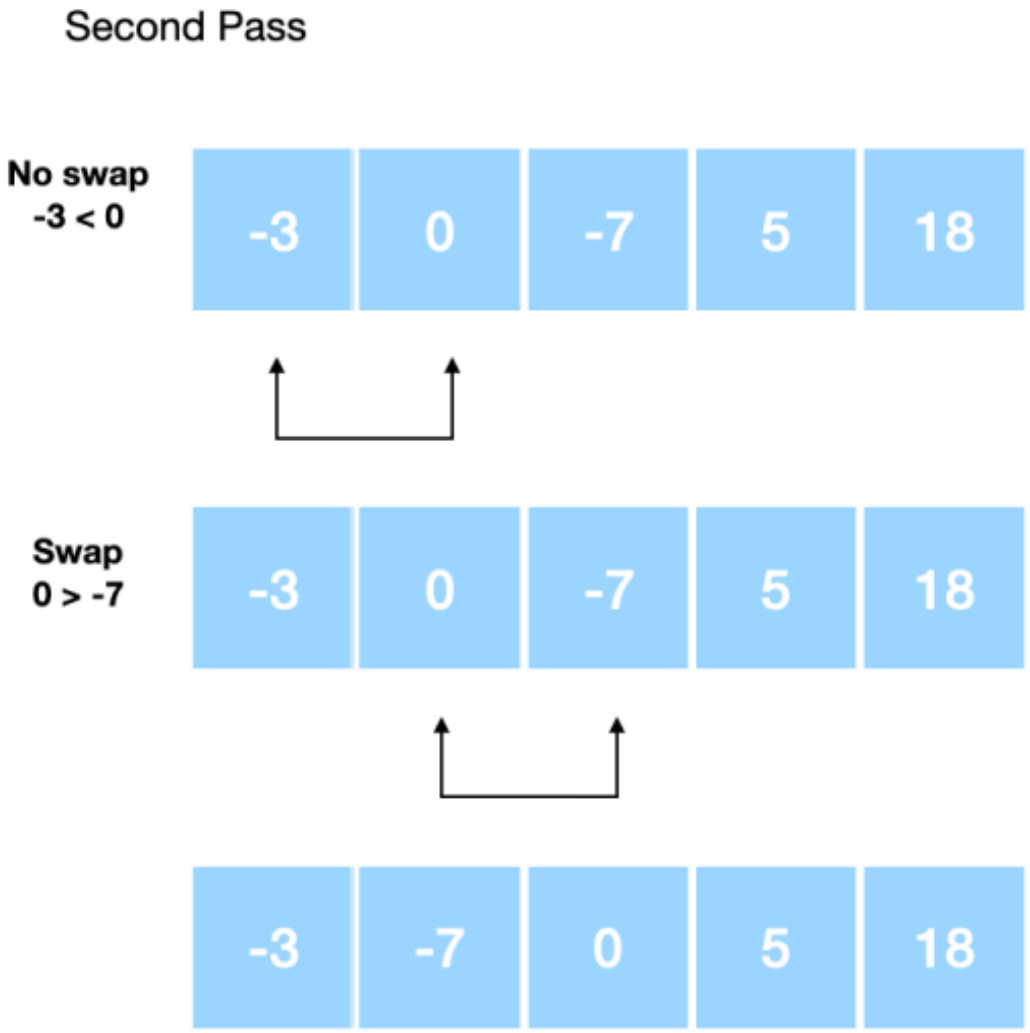
First Iteration

- Starting from the first element in the array, compare the first and second elements.
- If the first element is greater than the second element, swap the elements.
- Compare the second and third elements, swap them if they are not in order.
- Proceed with the above process until the last element.



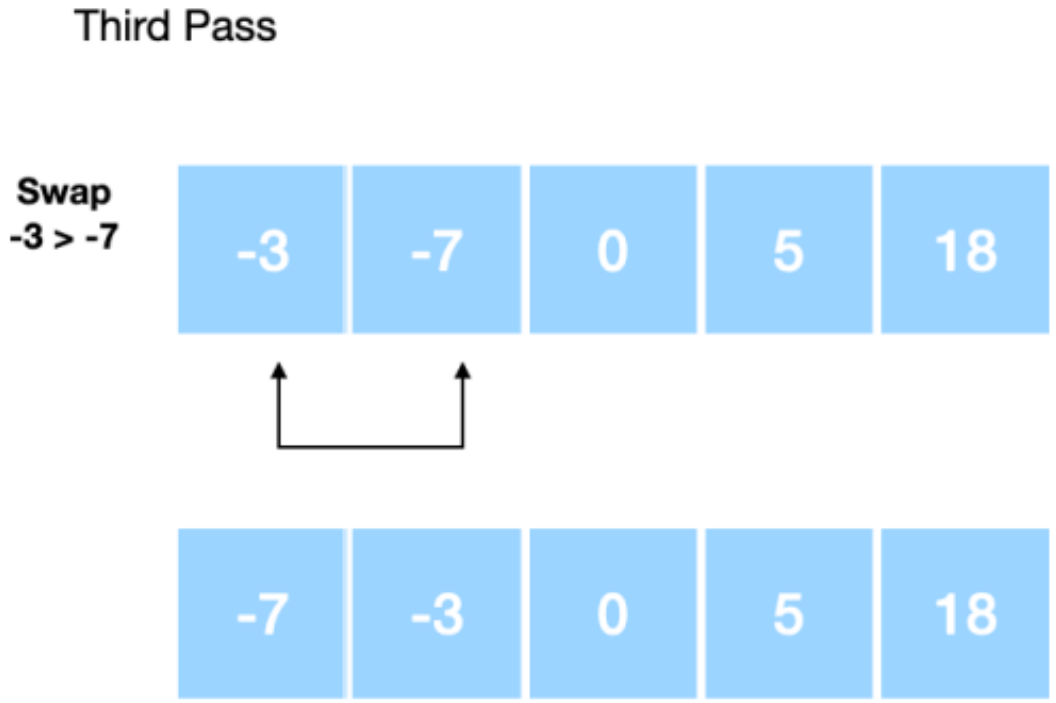
Second Iteration

- The same process goes on for the remaining iterations.
- After each iteration, the largest element among the unsorted elements is placed at the end.



Rest of the iterations

- The array is sorted when all the unsorted elements are placed at their correct positions.



Bubble Sort Algorithm

```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Bubble Sort Code

```
void bubbleSort(int array[], int size) {

  // loop to access each array element
  for (int step = 0; step < size; ++step) {

    // loop to compare array elements
    for (int i = 0; i < size - step; ++i) {

      // compare two adjacent elements
      // change > to < to sort in descending order
      if (array[i] > array[i + 1]) {

        // swapping elements if elements
        // are not in the intended order
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }
}
```

Time Complexities

- Worst Case Complexity: $O(n^2)$

  If we want to sort in ascending order and the array is in descending order then the worst case occurs.

- Best Case Complexity: O(n)

  If the array is already sorted, then there is no need for sorting.

- Average Case Complexity: $O(n^2)$

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

Space Complexity

- Space complexity is O(1) because an extra variable is used for swapping.

Visualization of Sorting Algorithms

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html