Sorting Algorithms

MD. MUKTAR HOSSAIN

LECTURER

DEPT. OF CSE

VARENDRA UNIVERSITY

Sorting

Sorting is the process of arranging the elements of an array so that they can be placed either in ascending or descending order.

There are so many sorting algorithms and these are followings:

- Comparison Based:

✓Bubble Sort	✓Quick Sort
✓Selection Sort	✓Heap Sort
✓Insertion Sort	✓Cycle Sort
✓Merge Sort	✓3-way Merge Sort

- Non Comparison Based:

✓Counting Sort	✓Bucket Sort
✓Radix Sort	✓Pigeonhole Sort

- Hybrid Sorting Algorithms:

✓IntroSort
✓Tim Sort

Applications of Sorting

- **Quickly Finding:** Once we sort the array, we can find k-th smallest and k-th largest elements in O(1) time for different values of k.
- **Searching Algorithms:** Sorting is often a crucial step in search algorithms like binary search, Ternary Search, where the data needs to be sorted before searching for a specific element.
- **Data management:** Sorting data makes it easier to search, retrieve, and analyze.
- **Database optimization:** Sorting data in databases improves query performance. We typically keep the data sorted by primary index so that we can do quick queries.
- **Machine learning:** Sorting is used to prepare data for training machine learning models.
- **Data Analysis:** Sorting helps in identifying patterns, trends, and outliers in datasets. It plays a vital role in statistical analysis, financial modeling, and other data-driven fields.
- **Operating Systems:** Sorting algorithms are used in operating systems for tasks

Advantages of Sorting

- **Efficiency:** Sorting algorithms help in arranging data in a specific order, making it easier and faster to search, retrieve, and analyze information.
- **Improved Performance:** By organizing data in a sorted manner, algorithms can perform operations more efficiently, leading to improved performance in various applications.
- **Simplified data analysis:** Sorting makes it easier to identify patterns and trends in data.
- **Reduced memory consumption:** Sorting can help reduce memory usage by eliminating duplicate elements.
- **Improved data visualization:** Sorted data can be visualized more effectively in charts and graphs.

Bubble Sort Algorithm

- Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order.
- **Algorithm:**
  - Compare the first two elements. If the first element is larger than the second, swap them.
  - Move to the next pair of adjacent elements and repeat the comparison and possible swap.
  - Continue this process for the entire list. After the first pass, the largest element will have "bubbled" up to the end of the list.
  - Repeat the process for the remaining unsorted portion of the list (ignoring the last sorted elements).
  - The algorithm continues until no swaps are needed, indicating that the list is sorted.
- **Time Complexity:**
  - Worst-case time complexity: $O(n^2)$ — when the list is in reverse order.
  - Best-case time complexity: $O(n)$ — when the list is already sorted
  - Average-case time complexity: $O(n^2)$.

Bubble Sort Cont'd

Let the elements of array are –

15	12	26	4	18

First Pass (i=0)

15(j)	12(j+1)	26	4	18
12	15(j)	26(j+1)	4	18
12	15	26(j)	4(j+1)	18

12	15	4	26(j)	18(j+1)
12	15	4	18	26

```
void bubbleSort (int arr[], int n){
    int i, j;
    bool swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
                swapped = true;
            }
        }
if (swapped == false)
            break;
    }
}
```

Bubble Sort Cont’d

12	4	15	18	26

Second Pass (i=1)

12(j)	15(j+1)	4	18	26
12	15(j)	4(j+1)	18	26

12	4	15(j)	18(j+1)	26
12	4	15	18	26

```
void bubbleSort (int arr[], int n){
   int i, j;
   bool swapped;
   for (i = 0; i < n - 1; i++) {
      swapped = false;
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            swap(&arr[j], &arr[j + 1]);
            swapped = true;
         }
      }
if (swapped == false)
         break;
   }
}
```

Bubble Sort Cont’d

12	4	15	18	26

Third Pass (i=2)

12(j)	4(j+1)	15	18	26

4	12(j)	15(j+1)	18	26
4	12	15	18	26

```
void bubbleSort (int arr[], int n){
   int i, j;
   bool swapped;
   for (i = 0; i < n - 1; i++) {
      swapped = false;
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            swap(&arr[j], &arr[j + 1]);
            swapped = true;
         }
      }
if (swapped == false)
         break;
   }
}
```

Bubble Sort Cont'd

4	12	15	18	26

Fourth Pass (i=3)

4(j)	12(j+1)	15	18	26
4	12	15	18	26

```
void bubbleSort (int arr[], int n){
   int i, j;
   bool swapped;
   for (i = 0; i < n - 1; i++) {
      swapped = false;
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            swap(&arr[j], &arr[j + 1]);
            swapped = true;
         }
      }
if (swapped == false)
         break;
   }
}
```

Insertion Sort Algorithm

- Insertion sort is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
- **Algorithm:**
  - Start from the second element of the list.
  - Compare the current element with the previous elements of the sorted section.
  - Move all the elements that are greater than the current element one position to the right to make space for the current element.
  - Insert the current element into the correct position in the sorted section.
  - Repeat this process until all elements are processed.
- **Time Complexity:**
  - Worst-case time complexity: $O(n^2)$ — when the list is in reverse order.
  - Best-case time complexity: $O(n)$ — when the list is already sorted
  - Average-case time complexity: $O(n^2)$.

Insertion Sort Cont’d

Let the elements of array are –

15	12	26	4	18

Key = 12

15(j)	12	26	4	18

15	15	26	4	18

12	15	26	4	18

```
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;
   }
}
```

Insertion Sort Cont’d

Key = 26

🟩 12	🟨 15(j)	26	4	18

🟩 12	🟥 15	26	4	18

🟩 12	🟩 15	🟩 26	4	18

Key = 4

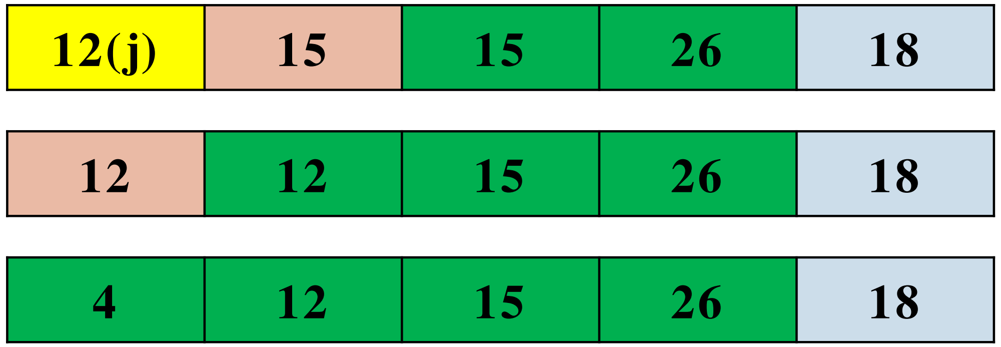
🟩 12	🟩 15	🟨 26(j)	4	18

🟩 12	🟨 15(j)	26	🟩 26	18

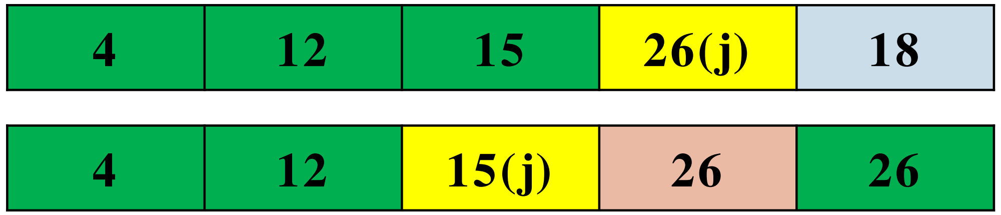
```
void insertionSort (int arr[], int n)
{
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;

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

Insertion Sort Cont'd

Key = 4



Key = 18



```
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;
   }
}
```

Insertion Sort Cont'd

Key = 18

4	**12**	**15(j)**	**26**	**26**

4	**12**	**15**	**18**	**26**

```
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;
   }
}
```

Selection Sort Algorithm

- Selection Sort works by repeatedly finding the smallest (or largest, depending on the order) element from the unsorted part of the list and swapping it with the first unsorted element..
- **Algorithm:**
  - Starting with the first element, look through the entire list to find the smallest element.
  - Swap the smallest element found with the first element.
  - Move to the next element and repeat the process for the remaining unsorted portion of the list.
  - Continue this process until the entire list is sorted.
- **Time Complexity:** Selection sort performs the same number of comparisons regardless of the input order
  - Worst-case time complexity: $O(n^2)$.
  - Best-case time complexity: $O(n^2)$.
  - Average-case time complexity: $O(n^2)$.

Selection Sort Cont’d

Let the elements of array are –

15	12	26	4	18

15(i)	12	26	4(min)	18
4	12(i) (min)	26	15	18
4	12	26(i)	15(min)	18

4	12	15	26(i)	18(min)

4	12	15	18	26

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

Heap Sort Algorithm

Home Task

Link 1

Link 2

Thank You