WOLLO UNIVERSITY
KOMBOLCHA INSTITUTES OF TECHNOLOGY
College of Informatics

Data Structures and Algorithms

Chapter 3

Simple Sorting and Searching Algorithms

Belachew N.
nbelay2112@gmail.com

Outline

- ✓ Sorting
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
- ✓ Searching
  - Linear/Sequential Searching
  - Binary Searching

Introduction to Sorting Algorithm

- ✓Sorting refers to arranging data in a particular format.
- ✓A sorting algorithm is an algorithm that puts elements of a list in a certain order.
- ✓The most used orders are numerical order.
- ✓Efficient sorting is important to optimize the use of other algorithms that require sorted lists to work correctly

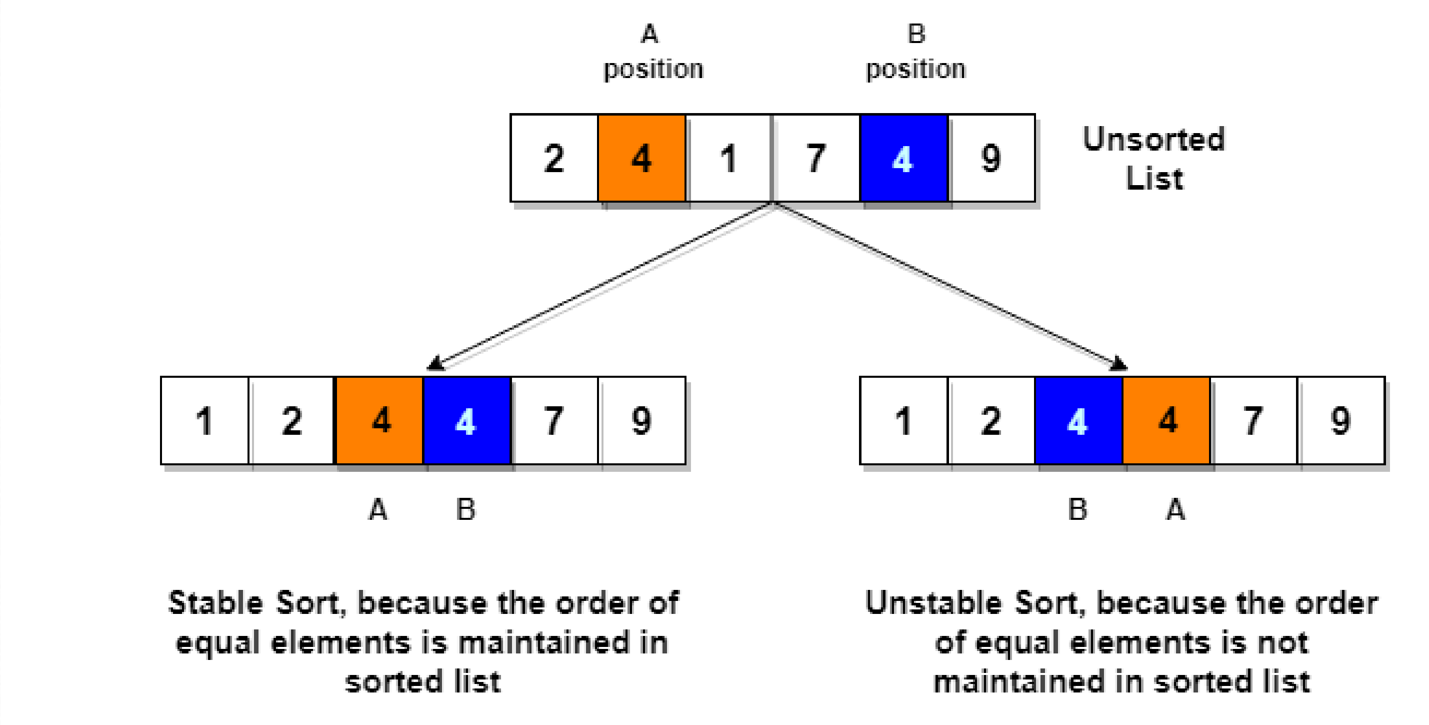
In-place Sorting and Not-in-place Sorting

- ✓Sorting algorithms may require some extra space for comparison and temporary storage of few data elements.
- ✓These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself. This is called in-place sorting.
- ✓Bubble sort, insertion sort, and selection sort are in-place sorting algorithms. Because only swapping of the element in the input array is required.
- ✓However, in some sorting algorithms, the program requires space which is more than or equal to the elements being sorted.
- ✓Sorting which uses equal or more space is called not-in-place sorting. Merge-sort is an example of not-in-place sorting.

Stable and Not Stable Sorting

- ✓If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called stable sorting.
- ✓A Stable Sort will guarantee that the original order of data having the same rank is preserved in the output.
- ✓Bubble sort, insertion sort and merge sort can be applying as stable algorithms
- ✓If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called unstable sorting.
- ✓Selection sort, heap sort and quick sort are an example of unstable sorting algorithms.

…cont’d



2	4	1	7	4	9

1	2	4	4	7	9

1	2	4	4	7	9

Stable Sort, because the order of equal elements is maintained in sorted list

Unstable Sort, because the order of equal elements is not maintained in sorted list

Simple Sorting Algorithms

✓ Simple sorting algorithms used to sort small-sized lists.

- Selection Sort
- Bubble Sort
- Insertion Sort

Selection Sort

- Loop through the array from i=0 to n-1.
- Select the smallest element in the array from i to n
- Swap this value with value at position i.
- ✓ This algorithm is not suitable for large data.

How it works

- ✓Step 1 – Set MIN to location 0
- ✓Step 2 – Search the minimum element in the list
- ✓Step 3 – Swap with value at location MIN
- ✓Step 4 – Increment MIN to point to next element
- ✓Step 5 – Repeat until list is sorted

Implementation

```
void selectionSort(int list[]){
    int i,j, smallest;
    for(i=0;i<list.length;i++){
      smallest=i;
      for(j=i+1;j<list.length;j++){
        if(list[j]<list[smallest])
          smallest=j;
      }//end of inner loop
    temp=list[smallest];
        list[smallest]=list[i];
        list[i]=temp;
          } //end of outer loop
      }//end of selection_sort
```

Complexity Analysis of Selection Sort

- ✓ Worst Case Time Complexity [Big-O]: $O(n^2)$
- ✓ Best Case Time Complexity [Big-omega]: $\Omega(n^2)$
- ✓ Average Time Complexity [Big-theta]: $\Theta(n^2)$

Bubble Sort

- ✓Bubble sort is the simplest algorithm to implement and the slowest algorithm on very large inputs.
- ✓It is also known as exchange sort.
- ✓Basic Idea:
  - Loop through array from i=0 to n and swap adjacent elements if they are out of order.
- ✓It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order.

How it works

- ✓ Following are the steps involved in bubble sort(for sorting a given array in ascending order):
  - ✓ Step 1: Starting with the first element(index = 0), compare the current element with the next element of the array.
  - ✓ Step 2: If the current element is greater than the next element of the array, swap them.
  - ✓ Step 3: If the current element is less than the next element, move to the next element.
  - ✓ Step 4: Repeat steps 1–3 until no more swaps are required

Implementation

```
void bubbleSort(int[] list){
      for(int i=0;i<list.length;i++){
        for(int j=1;j<(list.length-i);j++){
            if(list[j-1]>list[j]){
                   int temp=list[j-1];
                   list[j-1]=list[j];
                   list[j]=temp;
              }
       }
     }
  }
```

Complexity Analysis of Bubble Sort

- ✓Worst Case Time Complexity [Big-O]: O(n^2)
- ✓Best Case Time Complexity [Big-omega]: Ω(n)
- ✓Average Time Complexity [Big-theta]: Θ(n^2)

Insertion Sort

- ✓ The insertion sort works just like its name suggests - it inserts each item into its proper place in the final list.
- ✓ The simplest implementation of this requires two list structures - the source list and the list into which sorted items are inserted.
- ✓ To save memory, most implementations use an in-place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place.

…cont’d

- ✓It is efficient for smaller data sets, but very inefficient for larger lists.
- ✓Insertion Sort is adaptive, that means it reduces its total number of steps if a partially sorted array is provided as input, making it efficient.
- ✓It is better than Selection Sort and Bubble Sort algorithms.

How it works?

- ✓ The process involved in insertion sort is as follows

1. The left most value can be said to be sorted relative to itself. Thus, we don’t need to do anything.
2. Check to see if the second value is smaller than the first one. If it is, swap these two values. The first two values are now relatively sorted.
3. Next, we need to insert the third value in to the relatively sorted portion so that after insertion, the portion will still be relatively sorted.
4. Remove the third value first. Slide the second value to make room for insertion. Insert the value in the appropriate position.
5. Now the first three are relatively sorted.
6. Do the same for the remaining items in the list.

…cont’d

- ✓ Example

44	**55**	12	42	94	18	06	67
44	55	**12**	42	94	18	06	67
12	44	55	**42**	94	18	06	67
12	42	44	55	**94**	18	06	67
12	42	44	55	94	**18**	06	67
12	18	42	44	55	98	**06**	67
06	12	18	42	44	55	98	**67**
06	12	18	42	44	55	67	98

Implementation

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

Complexity Analysis of Insertion Sort

- Worst Case Time Complexity [Big-O]: O(n^2)
- Best Case Time Complexity [Big-omega]: Ω(n)
- Average Time Complexity [Big-theta]: Θ(n^2)

Introduction to Searching Algorithms

- ✓ Searching is an operation that helps finds the place of a given element or value in the list.
- ✓ The searching algorithms are used to search or find one or more than one element from a specific data structures.
- ✓ There are two simple searching algorithms.
  1. Linear (Sequential) Search, and
  2. Binary Search

Linear Search

- ✓Linear search is a very basic and simple search algorithm.
- ✓In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.
- ✓Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

How it works

- ✓ Step 1: Set i to 1
- ✓ Step 2: if i > n then go to step 7
- ✓ Step 3: if A[i] = x then go to step 6
- ✓ Step 4: Set i to i + 1
- ✓ Step 5: Go to Step 2
- ✓ Step 6: Print Element x Found at index i and go to step 8
- ✓ Step 7: Print element not found
- ✓ Step 8: Exit

Implementation

```
int linearSearch(int[] list, int key){
    for(int i=0;i<list.length;i++){
      if(list[i]==key){
        return i;
      }
    }
    return -1;
  }
```

Complexity Analysis of Linear Search

- **Best case-**

✓ The element being searched may be found at the first position.

✓ In this case, the search terminates in success with just one comparison.

✓ Thus in best case, linear search algorithm takes O(1) operations.

- **Worst Case-**

✓ The element being searched may be present at the last position or not present in the array at all.

✓ Thus in worst case, linear search algorithm takes O(n) operations.

✓ Time Complexity of Linear Search Algorithm is O(n).

✓ Here, n is the number of elements in the linear array.

Binary Search Algorithm

- ✓ Binary Search is applied on the sorted array or list of large size.
- ✓ The only limitation is that the array or list of elements must be sorted for the binary search algorithm to work on it.
- ✓ Binary Search is one of the fastest searching algorithms.
- ✓ It works on the principle of divide and conquer technique.

How it works

- We basically ignore half of the elements just after one comparison.
- Step 1: Compare x with the middle element.
- Step 2: If x matches with middle element, we return the mid index.
- Step 3: Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half and start with step 1
- Step 4: Else (x is smaller) then pick the elements to the left of the middle index, and start with Step 1.
- When a match is found, return the index of the element matched.
- If no match is found, then return -1

Implementation

```
void binarySearch(int[] arr, int first, int last, int key){
   int mid = (first + last)/2;
   while( first <= last ){
      if ( arr[mid] < key ){
        first = mid + 1;
      }else if ( arr[mid] == key ){
        System.out.println("Element is found at index: " + mid);
        break;
      }else{
         last = mid - 1;
      }
      mid = (first + last)/2;
   }
   if ( first > last ){
        System.out.println("Element is not found!");
     }
   }
```

Complexity Analysis of Binary Search

✓ Time Complexity of Binary Search Algorithm is $O(\log_2 n)$.

✓ It eliminates half of the list from further searching by using the result of each comparison.

✓ It indicates whether the element being searched is before or after the current position in the list.

✓ This information is used to narrow the search.

✓ For large lists of data, it works significantly better than linear search.

Review Questions

1. Write the main criteria's to judge which algorithms better than other algorithm
2. What is the difference between stable and not stable sorting algorithm?
3. How to Optimize Bubble Sort Algorithm? Show with example
4. Explain the algorithm for insertion sort and give a suitable example.
5. Discuss the difference between binary search and linear search algorithm with example

End of Ch.3

Questions, Ambiguities, Doubts, … ???