Sorting Algorithms

IIITS

Different Sorting Algorithms

- Simple sorts
  - Insertion sort
  - Selection sort
- Efficient sorts
  - Merge sort
  - Heapsort
  - Quicksort
- Bubble sort and variants
  - Bubble sort
  - Shell sort
  - Comb sort
- Distribution sort
  - Counting sort
  - Bucket sort
  - Radix sort

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Bubble Sort

```
void bubblesort(int arr[], int size)
{
    int i, j;
    for (i = 0;  i < size; i++)
    {
        for (j = 0; j < size - i; j++)
        {
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
        }
    }
}
```

Bubble Sort

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in **bold** are being compared. Three passes will be required.

First Pass

(**5** **1** 4 2 8) → (**1** **5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5** **4** 2 8) → (1 **4** **5** 2 8), Swap since 5 > 4
(1 4 **5** **2** 8) → (1 4 **2** **5** 8), Swap since 5 > 2
(1 4 2 **5** **8**) → (1 4 2 **5** **8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass

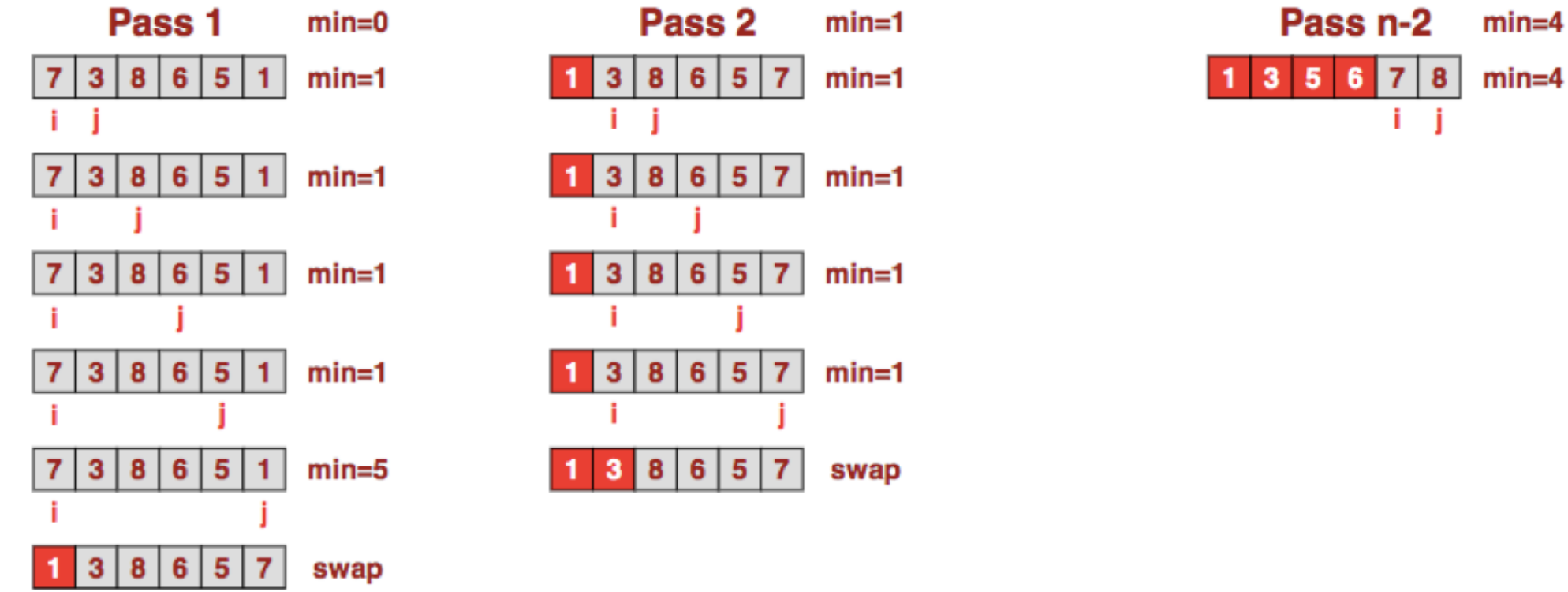
(**1** **4** 2 5 8) → (**1** **4** 2 5 8)
(1 **4** **2** 5 8) → (1 **2** **4** 5 8), Swap since 4 > 2
(1 2 **4** **5** 8) → (1 2 **4** **5** 8)
(1 2 4 **5** **8**) → (1 2 4 **5** **8**)
Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass

(**1** **2** 4 5 8) → (**1** **2** 4 5 8)
(1 **2** **4** 5 8) → (1 **2** **4** 5 8)
(1 2 **4** **5** 8) → (1 2 **4** **5** 8)
(1 2 4 **5** **8**) → (1 2 4 **5** **8**)

Selection Sort

```
void ssort(vector<int> mylist)
{
  for(i=0; i < mylist.size()-1; i++){
     int min = i;
     for(j=i+1; j < mylist.size; j++){
        if(mylist[j] < mylist[min]) {
           min = j
     }  }
     swap(mylist[i], mylist[min])
}
```



Selection Sort Complexity

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the minimum requires scanning *n* elements (taking *n-1* comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining *n-1* elements and so on. Therefore, the total number of comparisons is -

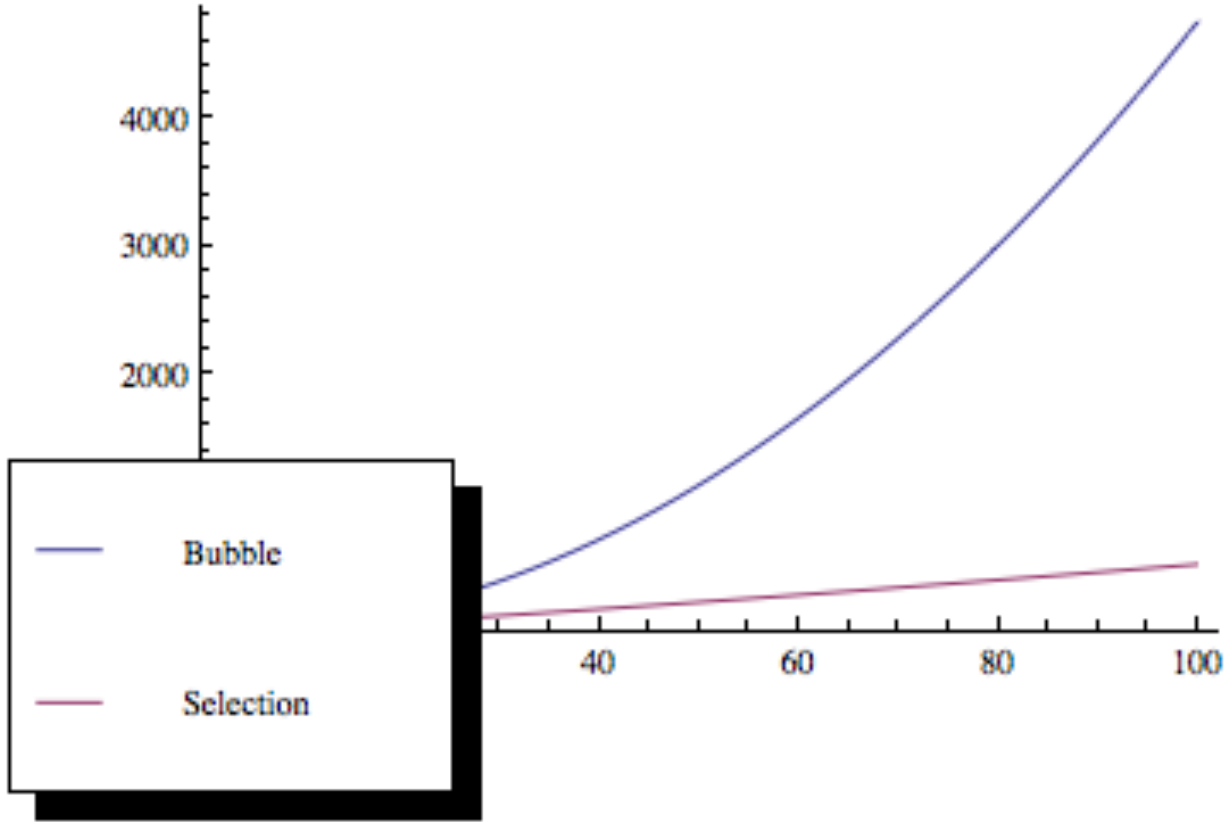
O(n^2) = $(n-1)+(n-2)+\ldots+1=\sum_{i=1}^{n-1} i=\sum_{i=1}^{n-1}\binom{i}{1}$

Selection sort almost always outperforms bubble sort! – not always true

- If the list is partially sorted.
- Selection Sort is used normally in cases where memory writes are quite expensive than memory reads. It only does O(n) memory writes, but Bubble Sort need at least twice that many memory writes.
- More specifically, Bubble sort requires, on average, n/4 swaps per entry (each entry is moved element-wise from its initial position to its final position, and each swap involves two entries), while Selection sort requires only 1 (once the minimum/maximum has been found, it is swapped once to the end of the array).

Bubble vs. Selection Sort

“Knuth's Art of Computer Programming” by Donald Knuth



*** This analysis is based on the assumption that the input is random - which might not be true all the time.**

Assignment

- Implement both the algorithms using linked-list.
- Take random input – varying the number of elements and plot number of items vs. time.