جامـــــعة المـــــستـقبـل
AL MUSTAQBAL UNIVERSITY

قســـم الامـــــن الـــــــــسيبرانـــــــي

Department of Cyber Security

Subject:

Searching And Sorting Algorithms

Class:

Second

Lecturer:

Asst. Prof. Dr. Ali Kadhum Al-Quraby

Lecture: (2)

Sorting Methods

Bubble Sort

The bubble sort is easy to understand and program. The basic idea of bubble sort is to pass through the file sequentially several times. In each pass, we compare each element in the file with its successor i.e., X[i] with X[i+1] and interchange two element when they are not in proper order. We will illustrate this sorting technique by taking a specific example. Bubble sort is also called as exchange sort.

❖ Example:

Consider the array x[n] which is stored in memory as shown below:

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]
33	44	22	11	66	55

Suppose we want our array to be stored in ascending order. Then we pass through the array 5 times as described below:

Pass 1: (first element is compared with all other elements).

We compare X[i] and X[i+1] for i = 0, 1, 2, 3, and 4, and interchange **X[i]** and **X[i+1]** **if X[i] > X[i+1]**. The process is shown below:

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	Remarks
33	44	22	11	66	55	
	22	44				
		11	44			
			44	66		
				55	66	
33	22	11	44	55	66	

The biggest number 66 is moved to (bubbled up) the right most position in the array.

Pass 2: (second element is compared).

i.e., we compare **X[i]** with **X[i+1]** for i=0, 1, 2, and 3 and interchange **X[i]** and **X[i+1]** **if X[i] > X[i+1]**. The process is shown below:

X[0]	X[1]	X[2]	X[3]	X[4]	Remarks
33	22	11	44	55	
22	33				
	11	33			
		33	44		
			44	55	
22	11	33	44	55	

The second biggest number 55 is moved now to **X[4]**.

Pass 3: (third element is compared).

We repeat the same process, but this time we leave both **X[4]** and **X[5]**. By doing this, we move the third biggest number 44 to **X[3]**.

X[0]	X[1]	X[2]	X[3]	Remarks
22	11	33	44	
11	22			
	22	33		
		33	44	
11	22	33	44	

Pass 4: (fourth element is compared).

We repeat the process leaving **X[3], X[4],** and **X[5]**. By doing this, we move the fourth biggest number 33 to **X[2]**.

X[0]	X[1]	X[2]	Remarks
11	22	33	
11	22		
	22	33	

Pass 5: (fifth element is compared).

We repeat the process leaving **X[2], X[3], X[4],** and **X[5]**. By doing this, we move the fifth biggest number 22 to **X[1]**. At this time, we will have the smallest number 11 in **X[0]**. Thus, we see that we can sort the array of size 6 in 5 passes.

For an array of size n, we required (n-1) passes.

❖ Program for Bubble Sort

Bubble Sort

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

// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        bool swapped = false; // Optimization: Track if a swap occurred
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        // If no swaps occurred, the array is already sorted
        if (!swapped)
            break;
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Main function
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Unsorted array: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

❖ Time Complexity:

The bubble sort method of sorting an array of size n requires **(n-1)** passes and **(n-1)** comparisons on each pass. Thus, the total number of comparisons is **(n-1) * (n-1) = n^2 - 2n + 1**, which is **O(n^2)**. Therefore, bubble sort is very inefficient when there are more elements to sorting.

Selection Sort

Selection sort will not require no more than n-1 interchanges. Suppose **x** is an array of size **n** stored in memory. The selection sort algorithm first selects the smallest element in the array **x** and place it at array position 0; then it selects the next smallest element in the array **x** and place it at array position 1. It simply continues this procedure until it places the biggest element in the last position of the array.

The array is passed through (n-1) times and the smallest element is placed in its respective position in the array as detailed below:

Pass 1: Find the location ***j*** of the smallest element in the array **x [0], x[1], ... x[n-1]**, and then interchange **x[j]** with **x[0]**. Then **x[0]** is sorted.

Pass 2: Leave the first element and find the location j of the smallest element in the sub-array **x[1], x[2], ... x[n-1]**, and then interchange **x[1]** with **x[j]**. Then

x[0], x[1] are sorted.

Pass 3: Leave the first two elements and find the location j of the smallest element in the sub-array **x[2], x[3], ... x[n-1]**, and then interchange **x[2]** with **x[j]**. Then **x[0], x[1], x[2]** are sorted.

Pass (n-1): Find the location j of the smaller of the elements **x[n-2]** and **x[n-1]**, and then interchange **x[j]** and **x[n-2]**. Then **x[0], x[1], ... x[n-2]** are sorted. Of course, during this pass **x[n-1]** will be the biggest element and so the entire array is sorted.

❖ Time Complexity:

In general, we prefer selection sort in case where the insertion sort or the bubble sort requires exclusive swapping. In spite of superiority of the selection sort over bubble sort and the insertion sort (there is significant decrease in run time), its efficiency is also **$O(n^2)$** for n data items.

❖ Example:

Let us consider the following example with 9 elements to analyze selection Sort:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*Remarks*
65	70	75	80	50	60	55	85	45	find the first smallest element
i								j	swap a[i] & a[j]
45	70	75	80	50	60	55	85	65	find the second smallest element
	i			j					swap a[i] and a[j]
45	**50**	75	80	70	60	55	85	65	Find the third smallest element
		i				j			swap a[i] and a[j]
45	**50**	**55**	80	70	60	75	85	65	Find the fourth smallest element
			i		j				swap a[i] and a[j]
45	**50**	**55**	**60**	70	80	75	85	65	Find the fifth smallest element
				i				j	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	80	75	85	70	Find the sixth smallest element
					i			j	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	75	85	80	Find the seventh smallest element
						i j			swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	85	80	Find the eighth smallest element
							i	J	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	**80**	**85**	The outer loop ends.