Lecture # 19

Sorting Algorithms

Selection Sort

- The basic idea of Selection Sort is to make a number of passes through the list or part of the list and, on each pass, select one element to be correctly positioned. For example, on each pass through a sub list, the smallest element in the sub list might be found and then moved to its proper location.

- As an illustration, suppose that following list has to be sorted into ascending order :

67 33 21 84 49 50 75

- We scan the list to locate the smallest element and find it in position 3:
- Then we interchange this element with the first element and thus properly position the smallest element at the beginning of the list.

21 33 67 84 49 50 75

- We now scan the sub list consisting of the elements from position 2 on to find the smallest element

  21 33 67 84 49 50 75

  and exchange it with the second element (itself in this case) and then properly position the next-to-smallest element in position 2

- We continue in this fashion, locating the smallest element in the sub list of elements from position 3 and interchanging it with the third element, then properly positioning the smallest element in the sub list of elements from position 4 on, and so on until we eventually do this for the sub list consisting of the last two elements as follows.

21	33	**49**	84	67	50	75
21	33	49	**50**	67	84	75
21	33	49	50	**67**	84	75
21	33	49	50	67	**75**	84

- Positioning the smallest element in this last sub list obviously also positions the last element correctly and thus completes the sort.

Exchange Sort (Bubble Sort)

- Exchange sort systematically interchange pairs of elements that are out of order until eventually no such pairs remain and the list is therefore sorted.

- To illustrate bubble sort, consider the following list

67 33 21 84 49 50 75

- On the first pass, we compare the first two elements, 67 and 33 and interchange them because they are out of order.

67 33 21 84 49 50 75

33 67 21 84 49 50 75

- Now we compare the second and third elements, 67 and 21 and interchange them

33 21 67 84 49 50 75

- Next we compare 67 and 84 but do not interchange them because they are already in the correct order.

  33 21 67 84 49 50 75

- Next 84 and 49 are compared and interchanged.

  33 21 67 49 84 50 75

- Then 84 and 50 are compared and interchanged.

  33 21 67 49 50 84 75

- Finally 84 and 75 are compared and interchanged.

  33 21 67 49 50 75 84

So the first pass through the list is now complete.

- We are guaranteed that on this pass, the largest element in the list will go down to the end of the list, since it will obviously be moved past all smaller elements.
- But notice that some of smaller items have "Bubbled Up" toward their proper positions nearer the front of the list.
- So we scan the list again but this time we ignore the last item because it is already in its proper location.

33 21 67 49 50 75 **84**

So in each pass the last element will be placed in its proper location in the sub list.

- The *worst case* of bubble sort occurs when the list elements are in reverse order because in this case only one item (the largest) item is placed correctly on each pass through the list.

- On the first pass through the list n-1 comparisons and interchanges are made and only the largest element is correctly positioned.

- On the next pass, sub list consisting of the first n-1 elements are scanned; there are n-2 comparisons and interchanges; and the next largest element sink to position n-1.

- Finally, process continues ***until*** the sub list consisting of the first two elements is scanned and on this pass there is one comparison & interchange.
- Thus a total of

$$(n-1) + (n-2) + (n-3) + \ldots\ldots + 1 = n(n-1)/2$$
$$= n^2/2 - n/2$$

Comparisons and interchanges are required.

Merge Sort

The **Merge sort** algorithm closely follows the divide and conquer paradigm. It works as follows.

- ***Divide*** : Divide the n - element sequence to be sorted into sub sequences each of size n / 2 elements.
- ***Conquer*** : Sort the two sub sequences recursively using merge sort.
- ***Combine*** : Merge the two sorted sub sequences to produce another sorted list.

- The key operation of the merge sort algorithm is the merging of two sub sequences in the "combine" step.

- To perform merging, we use Merge(A, left, mid, right); function, where A is an array of elements and left, right and mid representing leftmost, rightmost and center indices of an array respectively.

- The above function assumes that the sub array A[left…..mid] and A[mid+1 …… right] are in sorted order where n = right - left + 1

sorted sequence
1 2 2 3 4 5 6 6
merge
2 4 5 6
1 2 3 6
merge
merge
2 5
4 6
1 3
2 6
merge
merge
merge
merge
5
2
4
6
1
3
2
6
initial sequence

Quick Sort

- QuickSort is a Divide and Conquer algorithm.
- Picks an element as pivot and partitions the given array around the picked pivot.
- There are many different versions of Quick Sort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot
3. Pick a random element as pivot.
4. Pick median as pivot.

Quick Sort

- The key process in quickSort is partition().
- Given an array and an element x of array as pivot:
  1. Put x at its correct position in sorted array.
  2. Put all smaller elements (smaller than x) before x.
  3. Put all greater elements (greater than x) after x.

Partitioning an array

The Initial Array, where the pivot has been marked.

The first two elements are each compared with the pivot (and they are "swapped" with themself).

The next two element are greater than the pivot so they remain where they are.

The 2 is smaller than the pivot, so it is swapped with the first element available.

The 7 is larger, so it remains where it is.

Finally, the pivot is swapped into the correct location.

The array has now been partitioned, and the index of the pivot can be returned.

Insertion Sort

- Simple sorting algorithm that sorts elements by shifting them one by one.
- Similar to the way we sort playing cards in our hands.

Example

9	2	7	5	1	4	3	6

Types of Search

Linear / Sequential Search

- Most fundamental search algorithm
- Input to linear search is a sequence (array) plus a target key we intend to search
- UNORDERED array – elements of the array are not in any order that would aid the search (random)

- Simple sequential search where we would check each element of the array and compare it with our key
- If a match is found then our search is successful
- If none of the items match the key value, then the key item is not present in the array.
- If the array has n items, then in the worst case, all items in the sequence will have to be checked against the key value for equality

Binary Search

- Search algorithm applied to ordered lists
- Uses the divide and conquer approach
- At each step, compare the key with the middle element of the array
- split the array into two halves – upper half and lower half.
- If key equals middle element algorithm is done

- If key < middle element – discard upper half of array
- If key > middle element – discard lower half of array
- Repeat this process
- If the entire array has been discarded using this method, then the element is not found.

Thank You……