Sorting!!

CS112 Recitation

Ashwin Haridas
ah1058@rutgers.edu

Original slides by: Hoai-An Nguyen

Insertion Sort

Algorithm

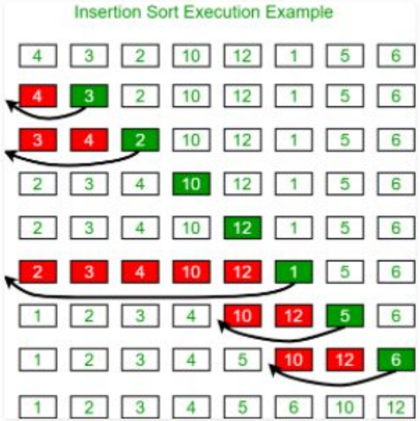
To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to its predecessor.

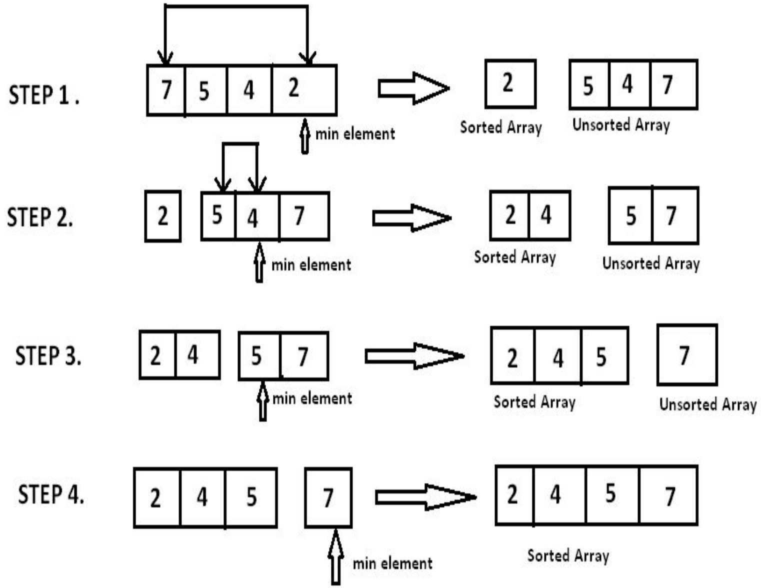
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Example:



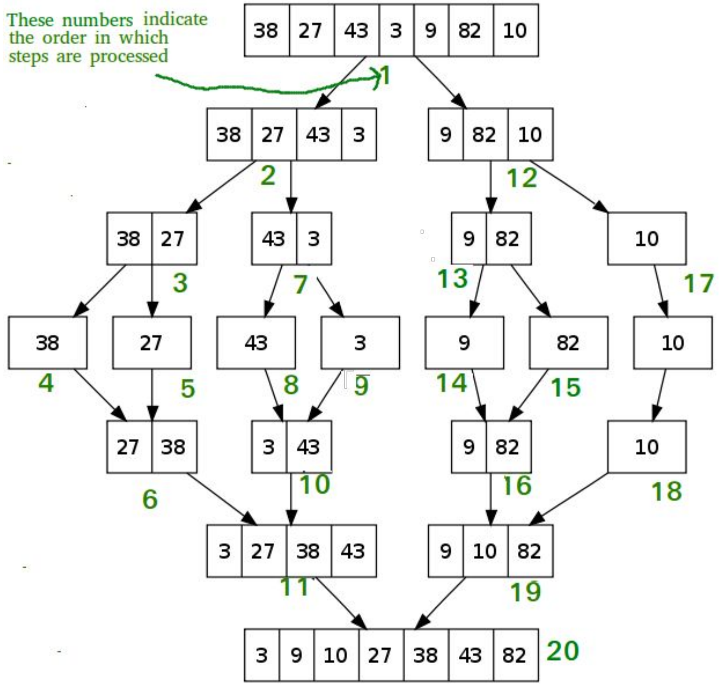
Selection Sort

```
void selection_sort (int A[ ], int n) {
        // temporary variable to store the position of minimum element

        int minimum;
        // reduces the effective size of the array by one in  each iteration.

        for(int i = 0; i < n-1 ; i++)  {

           // assuming the first element to be the minimum of the unsorted array .
             minimum = i ;

          // gives the effective size of the unsorted  array .

            for(int j = i+1; j < n ; j++ ) {
                if(A[ j ] < A[ minimum ])  {                 //finds the minimum element
                minimum = j ;
                }
            }
          // putting minimum element on its proper position.
          swap ( A[ minimum ], A[ i ]) ;
        }
   }
```



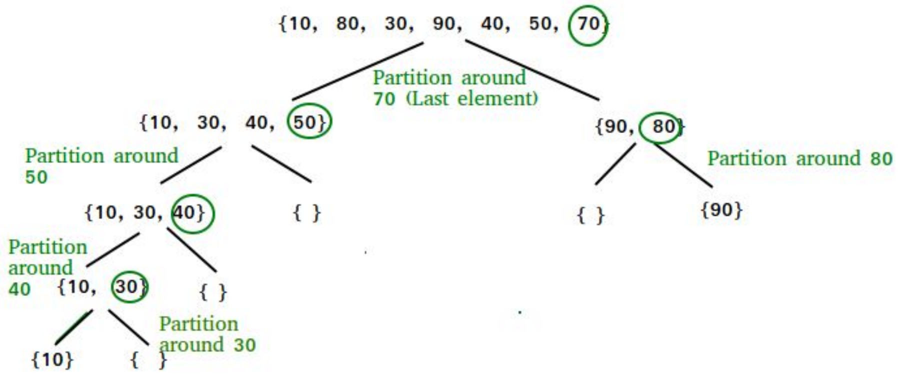
Merge Sort

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = l+ (r-l)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```



Quick Sort

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



Let’s Review

- **Insertion sort**: Insert the current item by moving larger items one position to the right before inserting the current item into the vacated position items
- **Selection sort**: Find the first unsorted item in the array and exchange it with the first entry, then do the same with the second, third, etc.
- **Mergesort**: Divide an array into two halves, recursively sort the halves, and merge the results
- **Quicksort**: Partition array into two subarrays, sort the subarrays, and combine the ordered subarrays (for mergesort, the array is divided in half; for quicksort, the position of the partition depends on the contents of the array)

Warm-Up

1. A stable sorting algorithm preserves the initial order of duplicate elements. Which of the following algorithms are stable?
   a. Insertion sort
   b. Mergesort
   c. Quicksort
2. About how many comparisons will quicksort make when sorting an array of *n* items that are all equal?

Warm-Up

1. A stable sorting algorithm preserves the initial order of duplicate elements. Which of the following algorithms are stable?
   a. **Insertion sort**
   b. **Mergesort**
   c. Quicksort (the partition algorithm does not guarantee that duplicate elements will stay in the same order)
2. About how many comparisons will quicksort make when sorting an array of *n* items that are all equal?
   a. When sorting an array of n items that are all equal, quicksort will make approximately nlog(n) comparisons. Each partition will divide the array in half, plus or minus one.

Problem 1: Heapsort

Given the following input array:
3, 26, 67, 25, 9, -6, 43, 82, 10, 54

1. Trace the linear time build-heap algorithm on this array, to build a max-heap. How many comparisons did it take to build the heap?
   a. Build-heap: start at index n/2,
   b. perform sink(), subtract 1 from n, repeat
      i. Once we are at the beginning, our array becomes a heap
2. Starting with this max-heap, show how the array is then sorted by repeatedly moving the maximum entry to the end and applying sift-down (sink) to restore the (remaining) heap. How many comparisons did it take to sort the heap?

Problem 1: Heapsort Solution

Array	Sink	Comparisons
3 26 67 25 9 -6 43 82 10 54	9	1
3 26 67 25 54 -6 43 82 10 9	25	2
3 26 67 82 54 -6 43 25 10 9	67	2
3 26 67 82 54 -6 43 25 10 9	26	4
3 82 67 26 54 -6 43 25 10 9	3	5
82 54 67 26 9 -6 43 25 10 3	done	

Problem 1: Heapsort Solution Cont.

Array	Sink	Comparisons
82 54 67 26 9 -6 43 25 10 3		
swap(82,3)		
3 54 67 26 9 -6 43 25 10 82	3	4
67 54 43 26 9 -6 3 25 10 82		
swap(67,10)		
10 54 43 26 9 -6 3 25 67 82	10	6
54 26 43 25 9 -6 3 10 67 82		
swap(54,10)		
10 26 43 25 9 -6 3 54 67 82	10	4
43 26 10 25 9 -6 3 54 67 82		
swap(43,3)		
3 26 10 25 9 -6 43 54 67 82	3	4

Problem 1: Heapsort Solution Cont.

26	25	10	3	9	-6	43	54	67	82		
swap(26,-6)											
-6	25	10	3	9	26	43	54	67	82	-6	4
25	9	10	3	-6	26	43	54	67	82		
swap(25,-6)											
-6	9	10	3	25	26	43	54	67	82	-6	2
10	9	-6	3	25	26	43	54	67	82		
swap(10,3)											
3	9	-6	10	25	26	43	54	67	82	3	2

Problem 1: Heapsort Solution Cont.

9	3	-6	10	25	26	43	54	67	82		
			swap(9,-6)								
-6	3	9	10	25	26	43	54	67	82	-6	1
3	-6	9	10	25	26	43	54	67	82		
			swap(3,-6)								
-6	3	9	10	25	26	43	54	67	82	done	

Problem 2: Partition

Show how the method partition() partitions the array

E A S Y Q U E S T I O N.

But before that… lets do a few examples of the partition() method

Problem 2: Partition

8 7 5 4 3 2 1

Problem 2: Partition

1 2 3 4 5 6 7 8

Problem 2: Partition

7 2 3 4 8 6 8 9

Problem 2: Partition

15 12 13 11 20 18 22 14

Problem 2: Partition

Show how the method partition() partitions the array

E A S Y Q U E S T I O N.

Problem 2: Partition Solution

Show how the method partition() partitions the array
E A S Y Q U E S T I O N.

Solution

		a[]											
i	j	0	1	2	3	4	5	6	7	8	9	10	11
		E	A	S	Y	Q	U	E	S	T	I	O	N
2	6	E	A	S	Y	Q	U	E	S	T	I	O	N
2	6	E	A	E	Y	Q	U	S	S	T	I	O	N
3	2	E	A	E	Y	Q	U	S	S	T	I	O	N
	2	E	A	E	Y	Q	U	S	S	T	I	O	N

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code:

Thanks for a great semester! Good luck on your exams, and have a good summer break :)

Survey