Week 1: Lab

1   Bubblesort

BUBBLE-SORT(A)
1   For $k = 1$ **to** $n - 1$
2   // do a bubble pass
3   For $i = 0$ **to** $n - 2$
4   if $A[i] > A[i + 1]$: swap

1. Run Bubblesort on $A = [3, 1, 5, 7, 4, 6, 2]$, and show the array after every iteration.

2. What can you say about the last element in A after one bubble pass? Make a statement and try to argue why it's true.

3. What must be true after two bubble passes? (Hint: there are two elements that are guaranteed to be in the right places...which ones?)

4. Using this insight, argue that after $n - 1$ bubble passes the input is sorted.

5. Because after $n - 1$ passes the input is guaranteed to be sorted (see above), this tells us that $n - 1$ bubble passes are *sufficient* to sort. Now we ask the following question: Are $n - 1$ passes sometimes necessary? Or put differently, are $n - 1$ passes necessary in the worst-case? Come up with an array A that needs precisely $n - 1$ bubble passes to be sorted.

6. What is the best case and worst case running time of Bubblesort, as written above? What sort of input arrays trigger these cases?

7. Write an improved version of BubbleSort that has an $O(n)$ best-case runing time. (Hint: avoid doing unnecessary bubble-passes.)

2 Selection Sort

SELECTION-SORT(A)

1   For $i = 0$ **to** (?)
2       $k =$ the index of the smallest element among $A[i], A[i+1], ..., A[n-1]$
3   swap $A[i]$ with $A[k]$

1. How many iterations of FindMin() (i.e. code on line 2))are necessary (best-case, worst-case) on an array of size n?

2. Show how Selection Sort works on the array $A = (3, 1, 5, 7, 4, 6, 2)$.

3. What is the best case and worst case running time of Selection sort? What sort of input arrays trigger these cases?

3 Insertion sort

INSERTION-SOR$T(A)$

1   For $k = 1$ **to** $n - 1$
2   //invariant: $A[0] \leq A[1] \leq A[2] \leq ...A[k-1]$
3       $key = A[k]$
4       $i = k - 1$
5   **while** $i \geq 0$ and $A[i] > key$
6           $A[i+1] = A[i]$
7           $i = i - 1$
8       $A[i+1] = key$

1. Show how Insertion sort works on $A = (3, 1, 5, 7, 4, 6, 2)$. The goal of this exercise is to go through the algorithm and understand how it works.

2. Note the *invariant* in line 2. An invariant is something that we claim to be true. In this case, the invariant is inside the loop so we claim it is true at at every iteration of the loop; sometimes it is refered to as a *loop invariant*.

   Spell out the invariant for the first iteration of the loop($k = 1$). Is it true?

   Spell out the invariant state for the second iteration of the loop($k = 2$). Is it true? Why?

3. For a given value of k, how many times does the inner while loop run, in the best case?

4. What is the best case running time of Insertion sort using big-oh notation, and what sort of input triggers it?

5. Same question as above, but the worst-case.

4 Misc

1. **Counting comparisons:** As we'll continue the investigation of sorting algorithms, we'll ask the question: can we do better than quadratic time? and more generally, we'll ask: What is the (worst-case of the) best possible sorting algorithm? This is called a lower bound for sorting. It turns out that establishig a lower bound is deeply connected to the power of the instructions used by the algorithm. More on this later, but for now we just want to notice that all of bubble sort, selection sort and insertion sort use only *comparisons between the elements of the array* to figure out order. These type of algorithms are refered to as *comparison-based sorting*. We'll see that it is possible to sort using something other than comparisons. More on that later.

   For each of the algorithms discussed above, identify the place(s) where they perform comparisons between array elements (underline or circle), and then answer the quetsions below:

   (a) Consider the following array $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

   How many *element comparisons* are performed by SelectionSort when run on A?

   (b) How many *element comparisons* are performed by InsertionSort when run on A?

   (c) How many element comparisons are performed by SelectionSort when run on $\{5, 4, 3, 2, 1\}$?

   (d) How many element comparisons are performed by InsertionSort when run on $\{5, 4, 3, 2, 1\}$?

   Note: Counting comparisons is important because it gets you used to thinking about the complexity of sorting as being the same, up to a constant factor, to the number of comparisons performed by a sorting algorithm. We'll use this insight later to show a lower-bound for sorting by showing a lower-bound on the number of comparison performed by a sorting algorithm.