Week 4/5 Review Sheet (Algorithms)

Concept and Tools

1. **Definition of Algorithms** – an ordered set of executable steps to accomplish a task
2. **Evaluate an Algorithm**
   a) Simplicity, elegance, readability…- Matters, but not that important/commonly used.
   b) Space Complexity – Important, but we are not focusing on it.
   c) Time Complexity – Very important! An intuitive understanding: one "step" (e.g. comparison, multiplication, …) takes one "time unit", how many "steps" means how many "time unit" the program will take, which is the time complexity.
      i. The Big-O Notation – Used to measure time complexity.
         1. How we define: $f(x) = O(g(x))$ if $\exists\, M\ and\ x_0,\ s.t. |f(x)| \leq M * g(x)\ for\ \forall x \geq x_0$
         2. Rank the Complexity Classes: Constant < Logarithmic < Linear < Log-Linear < Polynomial < Exponential (e.g. O(1) < O(log n) < O(n) < O(n log n) < O(n^k) < O(C^n))
         3. How do we decide the complexity (in an intuitive way): Ignore low-class components and coefficient. (e.g. if an algorithm takes $3n^2 + 2logn + 8n + 1$ steps, then the complexity is O (n^2)

Algorithm Examples

3. **Sorting**
   a) Bubble Sort – Works like bubbles
      i. In a role, compare each two elements that are next to each other (from left to right), swap the larger one to the right side. Do $n - 1$ roles.
      ii. How do we optimize:
         1. If no swap happens in a role, then stop (indicate the array is already sorted).
         2. In the x role, we do not have to check the last $x - 1$ element(s).
      iii. Time Complexity: O (n^2)
   b) Merge Sort – Divide and conquer
      i. First, cut array to two parts with about equivalent length, sort each of them using merge sort (recurrence). Then, merge the two sorted list (we have two cursors starting from the first index of the two lists, each time compare the two numbers the cursors is pointing at, pop out the smaller element and append it to the large (merged) list, and move its cursor to the next (right) element. Continue do the same thing until all elements are in the large list), and the whole array is sorted.
      ii. Time Complexity: O (n log n)
   c) Other Sort (Radix Sort, Insertion Sort, Selection Sort, Quick Sort, Heap Sort, …)
      i. See visualization at https://visualgo.net/bn/sorting
4. **Searching**
   a) Binary Search
      i. The array should be already sorted
      ii. Each time select the element at the middle of the array, and check the target is in the left or the right, then reduce the scale of the problem to n/2
      iii. Time Complexity: O (log n)
5. **Other Algorithms: Greedy, DP, …**