CHAPTER 2 – PROBLEMS

Problem 1. a. Give an example of an algorithm that should not be considered an application of the brute-force approach. DYNAMIC PROGRAMMING
b. Give an example of a problem that cannot be solved by a brute-force algorithm.

Yes, because the idea of selection sort involves iterating through the list, removing one element and find the location that it belongs to in sorted list and insert in there. It continues unitil no input elements remain

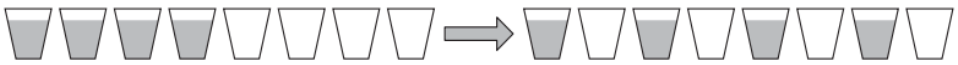
Problem 2. Is it possible to implement selection sort for linked lists with the same $O(n^2)$ efficiency as the array version?

Problem 3. Alternating disks You have a row of 2n disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks.

Design an algorithm for solving this puzzle and determine the number of moves it takes. Does your algorithm follow a brute-force approach?

Problem 4. Alternating glasses.

a. There are 2n glasses standing next to each other in a row, the first n of them filled with a soda drink and the remaining n glasses empty. Make the glasses alternate in a filled-empty-filled-empty pattern in the minimum number of glass moves.



b. Solve the same problem if 2n glasses—n with a drink and n empty—are initially in a random order.

Problem 5. Compare the slide's implementation of insertion sort with the following version.

```
ALGORITHM INSERTION-SORT2(A[0…n-1])
  for i ← 1 to n – 1 do
      j ← i – 1
          while j ≥ 0 and A[j] > A[j+1] do
              swap(A[j], A[j+1])
              j ← j – 1
```

What is the time efficiency of this algorithm? How is it compared to that of the version given in the slide?

Problem 6. Write pseudocode for a divide-and-conquer algorithm for finding the position (index) of the largest element in an array of *n* numbers. How does this algorithm compare with the brute-force algorithm for this problem?

Problem 7. Find the order of growth for solutions of the following recurrences.

a. $T(n) = 4T(n/2) + n, T(1) = 1$

b. $T(n) = 4T(n/2) + n^2, T(1) = 1$

c. $T(n) = 4T(n/2) + n^3, T(1) = 1$

Problem 8. Give an example showing that quicksort is not a stable sorting algorithm.

Problem 9. What is the running time of Quicksort when all elements of array A have the same value?

Problem 10. Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \dots, n - 1$.

Problem 11. Answer the following questions.

a. Construct a heap for the list 1, 8, 6, 5, 3, 7, 4 by the bottom-up algorithm (in the slide).

b. Construct a heap for the list 1, 8, 6, 5, 3, 7, 4 by successive key insertions (design a top-down algorithm by yourself).

c. Is it always true that the bottom-up and top-down algorithms yield the same heap for the same input?

Problem 12. Suppose that we were to rewrite the for loop in line 7 of the Counting-Sort in the slide as

7. for i ← 0 to n – 1

Show that the algorithm still works properly, but that is not stable.

Problem 13. Describe an algorithm that, given n integers in the range *0* to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a : b]$ in *O(1)* time. Your algorithm should use $O(n + k)$ preprocessing time.