1. [15%] Design algorithms to compute a Fibonacci sequence as follows.
(a) Write a pseudo code to recursively compute Fibonacci(n), where n is a positive integer, by applying the divide and conquer approach. Ensure that your algorithm terminates and produces a correct Fibonacci sequence.

Answer: Slide 15 of the lecture note for dynamic programming.

(b) Draw a recursion tree for Fibonacci(4) using the algorithm you designed above.

Answer: Slide 17 of the lecture note for dynamic programming.

(c) Write a pseudo code to compute Fibonacci(n), where n is a positive integer, by applying the dynamic programming technique. Ensure that your algorithm terminates and produces a correct Fibonacci sequence.

Answer: Slide 18 of the lecture note for dynamic programming.

2. [15%][True/False Questions]. Just say T(ue) or F(alse) for the following questions:
   (a) $10^9 n^3 + n^2 + 3000n = \Theta(n^3) \rightarrow T$
   (b) $527n^2 = O(n^3) \rightarrow T$
   (c) $n^6 = o(n^4) \rightarrow F$
   (d) An optimal binary search tree can always be constructed by placing the data with the highest access probability at the root of a binary search tree. \( \rightarrow F \)
   (e) One should avoid divide and conquer, if a problem instance of size n is divided into two or more instances where the size of each instance is almost n. \( \rightarrow T \)

3. [10%] Answer the following questions.
   (a) What is the worst-case input to quicksort?
      Answer: A list of already sorted numbers.
   (b) Briefly describe how to avoid the worst case in quicksort.
      Answer: Randomize the pivot selection.
4. [10%] Prove that $a_1n + a_2n^2 + a_3n^3 + \ldots + a_k \cdot n^{k-1} + a_k \cdot n^k = \Theta(n^k)$ where $a_i > 0$ ($1 \leq i \leq k$).

Answer:
1) For $n \geq 1$, $a_1n + a_2n^2 + a_3n^3 + \ldots + a_k \cdot n^{k-1} + a_k \cdot n^k \leq k \cdot \max(a_1, a_2, \ldots, a_k) \cdot n^k$.
   Hence, $a_1n + a_2n^2 + a_3n^3 + \ldots + a_k \cdot n^{k-1} + a_k \cdot n^k = O(n^k)$.
2) For $n \geq 1$, $a_1n + a_2n^2 + a_3n^3 + \ldots + a_k \cdot n^{k-1} + a_k \cdot n^k \geq \min(a_1, a_2, \ldots, a_k) \cdot n^k$.
   Thus, $a_1n + a_2n^2 + a_3n^3 + \ldots + a_k \cdot n^{k-1} + a_k \cdot n^k = \Omega(n^k)$.

5. [10%] Prove that the principle of optimality does not hold for the longest simple path problem by giving a counter example. (Hint: A simple path does not visit a vertex more than once.)

Answer: Slide 13 of the lecture note for dynamic programming.

6. [10%] Answer the following questions.
   (a) Write recurrence equations for binary search performed in a sorted list of integers.

Answer:
   
   $T(1) = 1$
   $T(n) = 2T(n/2) + 1$

   (b) Solve the recurrence equation to compute the time complexity. Assume that the list has $n$ data items where $n = 2^k$ for a positive integer $k$. Show the computation procedure. Note that you will get no credit by simply giving an answer without showing the computation process.

Answer:
   
   $T(n) = T(n/2) + 1$
   $= T(n/4) + 2$
   $= T(n/8) + 3$
   $\ldots$
   $\ldots$
   $= T(1) + \lg n$
   $= \Theta(\lg n)$
7. [10%] Do the following.
(a) Sort the following list of data in non-descending order via radix sort. Show the sorting process in three steps, one step for each digit.

178, 139, 326, 572, 294, 321, 910, 368

Answer: Slide 18 of the lecture note for heap sort and radix sort.

(b) Radix sort has linear time complexity under a certain condition. Briefly describe the condition.

Answer: The range of the numbers to be sorted is bounded. Thus, an arbitrary number, e.g., a social security number, is expressed by a constant number of digits.

8. [10 points] Construct an optimal binary search tree for three keys 1, 2, and 3 that have search probabilities 0.2, 0.3, and 0.5, respectively.

Answer: See the next page.

9. [10 points] Use the Floyd’s algorithm to find the all pairs shortest paths in the following graph.

Answer: Slides 51 – 54 of the lecture note for dynamic programming.
1. $A[1][2] = \min_{1 \leq k \leq 3} (A[1][k-1] + A[k+1][2]) + 0.5$
   (a) $k=1$: $A[1][2] = A[1][0] + A[2][2] + 0.5 = 0.8$, $R[1][2] = 1$

2. $A[2][3] = \min_{2 \leq k \leq 3} (A[2][k-1] + A[k+1][3]) + 0.6$

3. $A[1][3] = \min_{1 \leq k \leq 3} (A[1][k-1] + A[k+1][0]) + 1$

You can choose either case 2 or case 3, because they have the same expected search cost. Let us pick case 2 here.