

**Due 23:59 Oct 5 (Sunday).** There are 100 points in this assignment.

Submit your answers (**must be typed**) in pdf file to CourSys

<https://coursys.sfu.ca/2025fa-cmpt-705-x1/>.

Submissions received after 23:59 will get penalty of reducing points: 20 and 50 points deductions for submissions received at  $[00 : 00, 00 : 10]$  and  $(00 : 10, 00 : 30]$  of Oct 6, respectively; no points will be given to submissions after 00 : 30 of Oct 6.

1. (Chapter 4 Problem 13 of the text book) 15 points

There is a set of jobs  $\{1, 2, \dots, n\}$  and a machine, each job  $i$  takes  $t_i$  time to complete by the machine. Given a schedule (i.e., an ordering of the jobs), let  $C_i$  denote the finishing time of job  $i$ . For example, if job  $j$  is the first to be done, then  $C_j = t_j$ ; and if job  $j$  is done right after job  $i$ , then  $C_j = C_i + t_j$ . Each job  $i$  also has a given weight  $w_i$ . Design an efficient algorithm which, given a set of  $n$  jobs with a processing time  $t_i$  and a weight  $w_i$  for each job  $i$ , orders the jobs so that the sum  $\sum_{i=1}^n w_i C_i$  is minimized, and analyze your algorithm.

2. 15 points

Given a set  $S = \{a_1, \dots, a_n\}$  of proposed jobs and  $p > 1$  resources, each job requires a resource and each resource can serve exactly one job at a time. Each job  $a_i$  has a start time  $s_i$  and a finish time  $f_i$  with  $0 \leq s_i < f_i < \infty$ . If job  $a_i$  is assigned to a resource  $r$ , then  $r$  serves  $a_i$  in time interval  $[s_i, f_i)$ . Jobs  $a_i$  and  $a_j$  are compatible if  $[s_i, f_i) \cap [s_j, f_j) = \emptyset$ . Design a greedy algorithm which assigns a maximum number of jobs from  $S$  to resources so that the jobs assigned to a same resource are mutually compatible, and analyze your algorithm.

3. (Chapter 5 Problem 2 of the text book) 15 points

Given a sequence of  $n$  distinct numbers  $a_1, \dots, a_n$ , a pair  $(a_i, a_j)$  is called a significant inversion if  $i < j$  and  $a_i > 2a_j$ . Give an  $O(n \log n)$  time algorithm to count the number of significant inversions for a given sequence of  $n$  distinct numbers, and analyze your algorithm.

4. (Chapter 5 Problem 3 of text book) 15 points

Given a set  $C$  of  $n$  cards, each card  $c_i \in C$  has a value  $v(c_i)$ . Two cards  $c_i$  and  $c_j$  are equivalent if  $v(c_i) = v(c_j)$  and the equivalence of two cards can be decided only by the direct comparison of the two cards. Give a divide-and-conquer algorithm which, given  $C$ , determines if there are more than  $n/2$  cards in  $C$  that all equivalent to one another using  $O(n \log n)$  direct comparisons of two cards, and analyze your algorithm.

5. (Chapter 6 Problem 1 of the text book) 15 points

A subset  $S$  of nodes in a graph  $G$  is an independent set if no two nodes in  $S$  is connected by an edge. Let  $G(V, E)$  be a path ( $V(G) = \{v_1, v_2, \dots, v_n\}$  and  $E(G) = \{\{v_i, v_{i+1}\}, 1 \leq i < n\}$ ). Each node  $v_i$  of  $G$  is assigned a positive integer weight  $w_i$ . For a subset  $S \subseteq V(G)$ , the weight of  $S$  is  $w(S) = \sum_{v_i \in S} w_i$ . Consider the problem of finding an independent set  $S$  of  $G$  such that  $w(S)$  is maximized.

(a) Give an example to show that the following "heaviest-first" greedy algorithm does not always find an independent set of maximum weight.

Start with  $S = \emptyset$

**while** there is a node in  $V(G) \setminus S$  **do**

    pick a node  $v_i$  of maximum weight and add  $v_i$  to  $S$

    delete  $v_i$  and its neighbors from  $V(G)$

**endwhile**

return  $S$

(b) Give an example to show that the following algorithm also does not always find an independent set of maximum weight.

Let  $S_1$  be the set of all  $v_i$ , where  $i$  is an odd number

Let  $S_2$  be the set of all  $v_i$ , where  $i$  is an even number

(Note that  $S_1$  and  $S_2$  are both independent sets)

Determine which of  $S_1$  or  $S_2$  has greater total weight, and return this one

(c) Give a dynamic programming algorithm (optimal solution structure, Bellman equation, pseudo code, and running time) that finds an independent set of  $G$  with maximum weight. The running time should be polynomial in  $n$ , independent of the values of the node weights.

6. (Chapter 6 Problem 5 of the text book) 15 points

Given a string of letters  $y = y_1y_2 \cdots y_n$ , segmentation of  $y$  is a partition of its letters into contiguous blocks of letters; each block corresponds to a word in the segmentation and has a quality value. The total quality of segmentation is the sum of the quality values of the words in the segmentation. Assume there is a black box  $\text{Qty}(x)$  which returns the quality value of a block  $x$ . Design a dynamic programming algorithm (optimal solution structure, Bellman equation, pseudo code and analysis of the algorithm) which, given  $y$  and  $\text{Qty}()$ , compute a segmentation with the maximum quality.

7. 10 points

Two dynamic programming algorithms (Knapsack and Knapsack1) for the knapsack problem are introduced in class. Implement the two algorithms and compare the space and time used by the algorithms. Note that the space efficiency of Algorithm Knapsack can be improved. Your answers to this question should include the the following results:

- The values of  $M[i, w]$  for each  $i = 0, 1, \dots, n$  and  $w = 0, 1, \dots, W$  for problem instance  $I = \{1, 2, 3, 4, 5\}$ ,  $\{v_1 = 3, v_2 = 6, v_3 = 18, v_4 = 22, v_5 = 28\}$ ,  $\{w_1 = 3, w_2 = 4, w_3 = 5, w_4 = 6, w_5 = 7\}$  and  $W = 11$  to show your program for Algorithm Knapsack solves the knapsack problem.
- The contents of  $A(j)$  for each  $j = 1, \dots, n$  for the same instance in (a) to show your program for Algorithm Knapsack1 solves the knapsack problem.
- For a few large problem instances, the memory space used by Algorithm Knapsack and the memory space used by Algorithm Knapsack1, a brief comparison of the results on the space used, the running time of the two algorithms and a brief comparison of the running times. The memory space can be either a measured maximum memory size or the maximum number of array elements used by the algorithms. You can randomly generate problem instances.

You may include comments to show your tricks to improve the memory space and running time of your implementations. You do not need to submit your program codes.