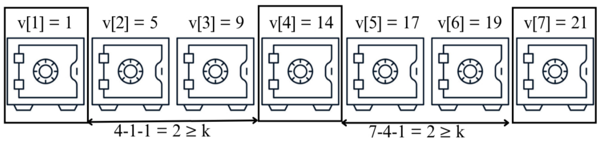
COP4533
Algorithm Abstraction & Design Programming Project

1 Problem Definition

Consider the problem of collecting treasure from a series of n vaults. Each vault i, where $1 \leq i \leq n$, has a value v_i. You may choose to include or skip each vault, but with the following restriction: whenever you pick a vault, you may not pick any other vault within k positions. In other words, if you choose vault i, then vaults $i-1, i-2, \ldots, i-k$ and vaults $i+1, i+2, \ldots, i+k$ are all excluded. You may not re-order the vaults. The total value of a particular arrangement is the total value of all the chosen vaults. Your goal is to determine a selection of vaults that maximize the total value.

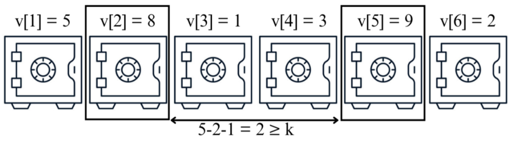
Example 1: $n = 7$, $k = 2$,
$v_i = [1, 5, 9, 14, 17, 19, 21]$

Solution : Pick $[v_1, v_4, v_7]$;
Total $= 1 + 14 + 21 = 36$



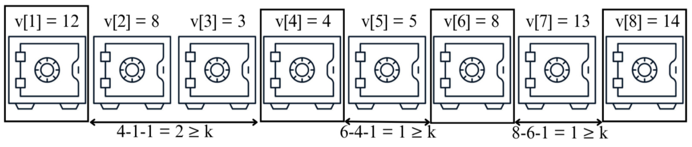
Example 2: $n = 6$, $k = 2$,
$v_i = [5, 8, 1, 3, 9, 2]$

Solution: Pick $[v_2, v_5]$;
Total $= 8 + 9 = 17$



Example 3: $n = 8$, $k = 1$,
$v_i = [12, 8, 3, 4, 5, 8, 13, 14]$

Solution: Pick $[v_1, v_4, v_6, v_8]$;
Total $= 12 + 4 + 8 + 14 = 38$



Below is the formal problem definition, starting with the general version, followed by two special cases. The special cases include input restrictions, which may allow for more efficient solutions.

ProblemG Given the values $v_1, \ldots, v_n$ of n vaults and a positive integer k, choose a subsequence of vaults such that no two chosen vaults are within k positions of each other, maximizing the total value.

ProblemS1 Given the values $v_1, \ldots, v_n$, where $v_i \leq v_j \; \forall i < j$, and a positive integer k, choose a subsequence of vaults such that no two chosen vaults are within k positions of each other, maximizing the total value.
(Note: The values form a monotonically non-decreasing sequence, as in Example 1.*)*

ProblemS2 Given the values $v_1, \ldots, v_n$, where $\exists p$ such that $\forall i < j \leq p$, $v_i \geq v_j$ and $\forall p \leq i < j$, $v_i \leq v_j$ and a positive integer k, choose a subsequence of vaults such that no two chosen vaults are within k positions of each other, maximizing the total value.
(Note: The values follow a unimodal function a single local minimum, as in Example 3.*)*

2 MILESTONE 1 - Greedy Algorithms

For the first part of the project, you will design, analyze and implement greedy algorithms to solve the special cases of the problem. You will conduct an experimental study on the performance of your algorithms. You are also required to construct examples showing how an algorithm developed for a special case of the problem might not work for the general version of the problem.

2.1 Design, Analysis, and Implementation Tasks

ALGORITHM1 Design a $\Theta(n)$ time greedy algorithm for solving PROBLEMS1.

ANALYSIS1 Prove the correctness of your greedy ALGORITHM1 for solving PROBLEMS1.

QUESTION1 Give an input example showing that Algorithm1 does not always solve PROBLEMG.

QUESTION2 Give an input example showing that Algorithm1 does not always solve PROBLEMS2.

PROGRAM1 Give an implementation of your greedy ALGORITHM1.

ALGORITHM2 Design a $\Theta(n)$ time greedy algorithm for solving PROBLEMS2.

ANALYSIS2 Prove the correctness of your greedy ALGORITHM2 for solving PROBLEMS2.

PROGRAM2 Give an implementation of your greedy ALGORITHM2.

2.2 Experimental Comparative Study

Test your implementations extensively for correctness and performance. For this purpose, you should create randomly generated input files of various sizes. The exact size of the experimental data sets that your program can handle depends on the quality of your implementation. For instance, you might want to choose $n = 10000, 20000, 30000, 40000, 50000$ to create at least five data sets for each experiment. Then, you should conduct and present a performance comparison of the following: For each comparison, generate a two dimensional plot of running time (y-axis) against input size (x-axis). These should be included in your report along with additional comments/observations. *Note: The size of your data sets should be such that the plots clearly illustrate how the running time of your algorithms grows with the input size*

PLOT1 Plot the running time of PROGRAM1 with respect to varying input size (as x-axis).

PLOT2 Plot the running time of PROGRAM2 with respect to varying input size (as x-axis).

2.3 Report

Prepare a report that describes design and analysis of your algorithms, presents the results of experimental study, and summarizes your learning experience. Your report should include but not limited to the following sections.

- **Team Members.** You may choose to work alone or in a team of up to three students on this assignment. If you choose to work in a team, clearly state the name and the contribution of each team member.
- **Algorithm Design and Analysis.** Give a clear description of your algorithm design and as well as its analysis. You can also include the pseudo code of your algorithms. Make sure to provide the answers to QUESTION1 and QUESTION2 as well.

- **Experimental Study.** Present the results of your experimental study. Include all your plots along with any explanation and comments you wish to provide.
- **Conclusion.** Summarize your learning experience on the first component of the project assignment. For each programming task, comment on the ease of implementation and other potential technical challenges.

2.4 Deliverables

The following contents are required for MILESTONE1 submission:

- **Implementation**: Submit the implementation code on Gradescope. Refer to Section 5 for more details. Make sure to include detailed comments next to each non-trivial block of code.
- **Report**: The report, in PDF format, must be submitted on Gradescope.

3 MILESTONE 2 - Dynamic Programming Algorithms

For the second part of the project, you will design, analyze and implement dynamic programming algorithms to solve the general cases of the problem. You will conduct an experimental study on the performance of your algorithms.

3.1 Design, Analysis, and Implementation Tasks

ALGORITHM3	Design a $\Theta(2^n)$ time naive algorithm for solving PROBLEMG.
ANALYSIS3	Give an analysis (correctness & running time) of ALGORITHM3 for solving PROBLEMG.
ALGORITHM4	Design a $\Theta(n^2)$ or $\Theta(n + (n-k)^2)$ time dynamic programming algorithm for solving PROBLEMG.
ANALYSIS4	Give an analysis (correctness & running time) of ALGORITHM4 for solving PROBLEMG.
ALGORITHM5	Design a $\Theta(n)$ time dynamic programming algorithm for solving PROBLEMG.
ANALYSIS5	Give an analysis (correctness & running time) of ALGORITHM5 for solving PROBLEMG.
PROGRAM3	Implementation of ALGORITHM3.
PROGRAM4A	Top-down recursive implementation of ALGORITHM4 using memoization.
PROGRAM4B	Iterative bottom-up implementation of ALGORITHM4.
PROGRAM5	Implementation of ALGORITHM5.

3.2 Experimental Comparative Study

Test your implementations extensively for correctness and performance. For this purpose, you should create randomly generated input files of various sizes. The exact size of the experimental data sets that your program can handle depends on the quality of your implementation. For instance, you might want to choose $n = 10000, 20000, 30000, 40000, 50000$ to create at least five data sets for each experiment. Then, you should conduct and present a performance comparison of the following: For each comparison, generate a two dimensional plot of running time (y-axis) against input size (x-axis). These should be included in your report along with additional comments/observations. *Note: The size of your data sets should be such that the plots clearly illustrate how the running time of your algorithms grows with the input size*

PLOT3 Plot the running time of PROGRAM3 with respect to varying input size (as x-axis).

PLOT4 Plot the running time of PROGRAM4A with respect to varying input size (as x-axis).

PLOT5 Plot the running time of PROGRAM4B with respect to varying input size (as x-axis).

PLOT6 Plot the running time of PROGRAM5 with respect to varying input size (as x-axis).

PLOT7 Overlay PLOTS 3,4,5,6 and contrasting the performance of PROGRAMS 3,4A, 4B,5.

PLOT8 Overlay PLOTS 4,5 and contrasting the performance of PROGRAMS 4A, 4B.

In addition to testing the running time performance of the algorithms described above, you are asked to conduct experiments to test the quality of the output of greedy ALGORITHM1, when it is run on the general case of the problem PROBLEMG. Note that such solution is not guaranteed to be optimal. Your task is to determine how different it is from the optimal. Make sure to conduct enough experiments with many random data sets. For x-axis, simply use n as the varying input size. On y-axis you can simply plot $(h_g - h_o)/h_o$, where h_o is the optimal height determined by any of PROGRAMS 3,4,5A,5B, and h_g is the height determined by ALGORITHM1.

PLOT9 Plot the output quality comparison $(h_g - h_o)/h_o$ of ALGORITHM1 and any of ALGORITHMS 3,4,5.

3.3 Report

Prepare a report that describes design and analysis of your algorithms, presents the results of experimental study, and summarizes your learning experience. Your report should include but not limited to the following sections.

- **Team Members.** You may choose to work alone or in a team of up to three students on this assignment. If you choose to work in a team, clearly state the name and the contribution of each team member.
- **Algorithm Design and Analysis.** Give a clear description of your algorithm design and as well as its analysis. Make sure to include the recursive formulation expressing optimal substructure for the dynamic programming algorithms. You can also include the pseudo code of your algorithms.
- **Experimental Study.** Present the results of your experimental study. Include all your plots along with any explanation and comments you wish to provide.
- **Conclusion.** Summarize your learning experience on the first component of the project assignment. For each programming task, comment on the ease of implementation and other potential technical challenges.

3.4 Deliverables

The following contents are required for MILESTONE2 submission:

- **Implementation**: Submit the implementation code on Gradescope. Refer to Section 5 for more details. Make sure to include detailed comments next to each non-trivial block of code.
- **Report**: The report, in PDF format, must be submitted on Gradescope along with source files.

4 MILESTONE 3 - Video Presentation

Prepare a video of length between 5 to 7 minutes, that presents all your results (design, analysis, experiments, and learning experience). We will provide more details regarding the submission and grading criteria on this component of the project.

5 Language/Input/Output Specifications

You may use C++, Java, or Python. Starter code for each language is provided (they differ only in filenames and class names). Add your implementation to the provided starter code in the language of your choice. Do **not** modify the filenames or class names in the given code. You must upload a file for each program on Gradescope (1 file per program). Do **not** include any extra files. You will receive instant feedback for a few example test cases to verify that your code follows the correct specifications.

The autograder on Gradescope runs on an Ubuntu 22.04 image. C++ code will be compiled using g++ 11.4.0 with the '-std=gnu++17' compile flag. Java code will be compiled and evaluated using OpenJDK 17. Python code will be evaluated using Python 3.10.

For convenience, you can assume that $1 \le k \le n < 10^5$, and $\forall i\ 1 \le v[i] < 10^5$. Test cases will be generated to accommodate the required time complexity. The generated test cases will also have a unique solution.

Input. Your program will read input from standard input (`stdin`) in the following order:

- Line 1 consists of two integers n and k separated by a single space.
- Line 2 consists of n integers (values of n vaults) separated by a single space.

Output. Your program should print to standard output (`stdout`) in the following order:

- Line 1 contains the optimal total value of the picked vaults.
- The next m lines contain the indices of the picked vaults in increasing order (1-indexed), where m is the number of vaults chosen.

6 Grading Policy

Milestones 1, 2, and 3 will weigh **30%**, **50%**, and **20%** of your total project grade, respectively. Outstanding submissions may receive up to **20%** bonus points.

For Milestones 1 and 2, grades will be based on the correctness & efficiency of algorithms and the quality of your experimental study and report:

- **Program 60%.** Correct/efficient design and implementation/execution. Also make sure to include comments with your code for clarity.
- **Report 40%.** Quality (clarity, details) of the write up on your design, analysis, programming experience, and experimental study.