- See previous practice problem sets for instructions.

1. Our neighbourhood machine shop has a buffing machine that they use to do the finishing touches on metal strips. A schematic of this machine is shown in Figure 1. The metal strip enters from the right side, gets processed by the machine, and exits from the left.

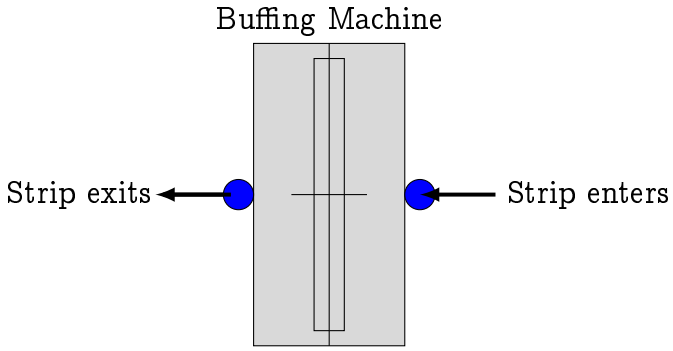

Figure 1: The buffing machine at work.

There is a maximum length M of continuous metal strip that this machine can process at one go. This length M is determined by various factors such as the length of the output tray and the heat and debris produced during the processing. Once the machine has processed length M worth of continuous metal strip the processed strip has to be cut out and removed, and the machine has to be stopped, cleaned, and cooled down before it can start processing again. Processing beyond length M at one go will result in very expensive recovery operations, so the machine shop has to avoid this at all costs.

The input strip comes with notches at random locations along its length, as shown in Figure 2. The machine has to cut the strip only at these notches, not at any other point along the strip.

Figure 2: Notches on the strip.

Once a metal strip is processed by the buffing machine, it has to be welded back together to its original length before being returned to the customer. To reduce the total cost of processing, the metal shop would like to minimize the number of times that each strip is cut by the buffing machine. Your task is to design a greedy algorithm that achieves *the minimum number of cuts*.

More formally: Let L be the length of an input strip, and let $x_1, x_2, \ldots, x_n$ be the distances—from the left end-point of the strip—at which the notches are located. We refer to each notch by its distance from the left end-point of the strip. We say that a subset N of these notches

is *feasible* if (i) the leftmost notch—say x_i—in N satisfies $x_i \leq M$, (ii) every two consecutive notches x_j, x_k in N satisfy $(x_k - x_j) \leq M$, and the rightmost notch—say x_ℓ—in N satisfies $(L - x_\ell) \leq M$. That is, a set of notches is feasible if and only if the buffing machine can process the entire strip by cutting it up exactly at these notches. We assume that the set of all available notches is feasible, since otherwise there is no way that the machine can process the entire strip without expensive stoppages. Your task is to design a greedy algorithm that finds a feasible set of notches of the *smallest size*.

I have described three greedy strategies below. In each case, either come up with an example to show that the strategy is not feasible/optimal, or prove that the strategy is optimal.

(a) **Very Greedy Strategy:** Whenever a notch is encountered, cut the strip at that notch.

(b) **Conservative Strategy:** Keep track of the length ℓ of strip that has been processed since the last cut. If no cut has been made so far, then ℓ is the length of the strip from the start of the strip to the current location. When a notch is encountered,

- If $\ell \geq \frac{M}{2}$ then cut the strip at the current notch
- If $\ell < \frac{M}{2}$ then don't cut the strip at the current notch

(c) **Optimistic Strategy:** When a notch is encountered, look ahead and find the distance ℓ of the *next* notch from the point where the last cut was made. If no cut has been made so far, then ℓ is the distance from the start of the strip to the *next* notch. If there is no next notch, then ℓ is the distance till the end of the strip.

- If $\ell > M$ then cut the strip at the current notch
- If $\ell \leq M$ then don't cut the strip at the current notch

2. To promote saving among poor households, the Government plans to issue a series of 12 bonds $B_1, B_2, \ldots, B_{12}$ on January 1, 2025. Each of these bonds has an initial issue price of ₹10,000/- and a duration of 12 months. The bonds accumulate compound interest on a monthly basis at *monthly* rates of $r_1 < r_2 < \cdots < r_{12}$ respectively where each $r_i > 1$. The maturity amount consisting of the principal and accumulated interest, will be paid out on January 1, 2026.

   To promote monthly savings, the Government (re)opens the bonds for subscription on the first day of each month till December 2025; that is, the bonds can be bought on Jan 1, Feb 1, ... Dec 1 2025. To be fair to all investors, the bonds are offered each month *at a price that includes the accumulated interest.* For instance: bond B_6 can be bought at ₹10,000/- on January 1, at ₹$10,000 \times r_6^3$ on April 1, and at ₹$10,000 \times r_6^{10}$ on November 1 (among other dates), and will be redeemed at ₹$10,000 \times r_6^{12}$ on January 1, 2026.

   To prevent hoarding, each person is allowed to buy *at most one unit* of each series. So a person—if they have the money—can buy at most one unit of each of the 12 bonds, where the buying is spread out over the 12 dates Jan 1, Feb 1, ... Dec 1 2025.

Suppose you have the money to buy any bond that is available for your purchase, at the prices that they are offered each month. What is a sequence of purchases that will maximize your total return (total redemption proceeds, minus total cost of purchase)? Use an *exchange argument* to prove that this sequence in fact maximizes the total return.

3. A certain school is organizing their annual cultural day. Students from the kindergarten (KG) will perform a colourful dance program. Each KG student wears an elaborate costume for the dance, with a tall and fancy hat. Each hat is so tall—and the kids are so small!—that it is made to order for each kid, after carefully measuring their heads and face so that the hat fits and the strap will hold the hat in place during the dance.

   Any two kids have different head and face sizes, so that each hat has to be worn by the kid for whom it was made. The tailor delivered the hats just in time on the day of the dance, but forgot to label them with the names of the kids to which they fit. So the KG teachers now have n kids and n hats to fit on the kids, but don't know which hat fits on which kid.

   The difference in sizes between any two kids is too small for visually comparing them and figuring out which of the two kids has a bigger head+face. Needless to say, the same holds for the hats: their sizes are so close that it is impossible to figure out visually, which of any two hats is bigger. The teachers *can* try putting a hat on a kid, and this will tell them if the kid is too small for the hat, is of the correct size for the hat, or is too large for the hat.

   There is not enough time to get a tape and measure the kids and/or the hats to get things sorted. Your job is to help the teachers figure out the right hat for each kid, by comparing hats to heads.

   (a) Describe—in words—a deterministic algorithm that will solve this problem with $\mathcal{O}(n^2)$ comparisons of hats with heads.

   (b) Describe—in words—a *randomized* algorithm that will solve this problem with $\mathcal{O}(n \log_2 n)$ *expected number* of comparisons of hats with heads.

   Hint: If both the hats and the kids are sorted in increasing order of size, we are done. Try to mimic Randomized Quicksort to make *both* these sorts happen.

4. As I mentioned in class, randomized algorithms are often (i) surprisingly simple compared to deterministic ones, and (ii) have better running times, at the cost of a small probability of failure. In this exercise, we will see an example of this where the deterministic algorithm is itself quite simple and as fast as can be hoped for, but the randomized algorithm manages to be even simpler and faster, and still has a probability of failure that we can choose to make as small as we want at a small extra expense in the running time.

   (a) Write the pseudocode for a **simple** *deterministic* algorithm that, given an array A of n integers, runs in $\mathcal{O}(n)$ time and returns an element x of A such that x is at least as large as the *median* element of A. Note that A may have repeated elements. For the sake of

simplicity you may assume that n is odd, so that the median element of A is uniquely defined: it is the one right in the middle of the sorted version of A.

Argue why your algorithm is correct, and why it runs in $\mathcal{O}(n)$ time.

(b) It turns out that we can beat the running time of this simple algorithm using randomization, if we are willing to be wrong with some small probability. Consider the following randomized algorithm for this task:

1. Choose a number $k < n$. Initialize an array $S[0, \ldots, (k-1)]$ to all NIL values.
2. For i in $0, 1, \ldots, (k-1)$ do:
   (a) Pick an index j of A uniformly at random.
   (b) Set $S[i] = A[j]$.
3. Find the maximum element x of S.
4. Return x.

What is the running time of this algorithm? What is the probability that it returns a correct answer? If you choose $k = 100$ in the first step, do these two values—running time, and probability of success—still look good for arbitrarily large values of n?

5. A set of points in the plane is said to be *collinear* if they all lie on a single line. Note that any set of points of size at most two is trivially collinear. In this exercise we will see how to use randomization to derive a fast algorithm that checks for a "large amount" of collinearity. With some—gory—imagination, you could think of this problem as modelling the question of where to place a gun in a 2D shooting game to get the maximum number of kills with one shot (assuming the bullet passes through opponents!).

   All points in this problem lie in the plane, and are given in the input as pairs of integers (x, y), denoting their X and Y coordinates respectively.

   (a) Write the pseudocode for a *deterministic* algorithm IsCollinear(p_1, p_2, p_3) that, given three points p_1, p_2, p_3 as input, runs in $\mathcal{O}(1)$ time and checks whether these three points are collinear. Explain why your algorithm is correct, and why it runs in $\mathcal{O}(1)$ time.

   (b) Write the pseudocode for a *deterministic* algorithm HasLargeHittableSet(p, S) that, given a point p and a set S of n points as inputs, checks whether there are *at least* $n/2$ distinct points $q_1, q_2, \ldots, q_t$; $t \geq n/2$ in S such that the set $\{p, q_1, q_2, \ldots, q_t\}$ is collinear. Your algorithm should run in $\mathcal{O}(n^2)$ time. Explain why your algorithm is correct, and why it runs in $\mathcal{O}(n^2)$ time.

   (c) Write the pseudocode for a *randomized* algorithm RandHasLargeHittableSet(p, S) with *one-sided error* that, given a point p and a set S of $n > 4$ points as inputs, checks whether there are *at least* $n/2$ distinct points $q_1, q_2, \ldots, q_t$; $t \geq n/2$ in S such that the set $\{p, q_1, q_2, \ldots, q_t\}$ is collinear.

Your algorithm should run in $\mathcal{O}(k \times n)$ and succeed with probability at least $1 - \frac{1}{2^k}$, for any positive integer k that you choose beforehand.

Explain why your algorithm has one-sided error, and why it runs within the stated running time bound and has the required probability of success.

6. For a positive integer k, an instance of the MAX kSAT problem—short for *Boolean Maximum k-Satisfiability*—is a Boolean formula in conjunctive normal form (CNF) with n variables and m clauses, where each clause contains exactly k literals[1]. The task is to find an assignment of truth values to the variables that satisfies as many clauses as possible.

   As an example, here is an input instance of the MAX 3SAT problem, with $n = 4$ variables and $m = 9$ clauses:

   $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee x_3 \vee x_4)$

   And here are three assignments that satisfy eight of these nine clauses, each:

   1. $\{x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 0\}$
   2. $\{x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 1\}$
   3. $\{x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1\}$

   So the solution to this instance is *at least* 8. I did an exhaustive search (using a program ...) and checked that there are no assignments that satisfy all the 9 clauses of this instance, so the solution to this instance of MAX 3SAT is exactly 8 (or my program has a bug).

   It so happens the MAX kSAT problem is *NP-hard* when k is at least 3. This means that we cannot hope for a polynomial-time algorithm that exactly solves every instance of MAX kSAT, when $k \geq 3$, in time that is polynomial in the size of the input formula. However, just like we saw for the MAX CUT problem, there is a simple randomized algorithm for MAX kSAT that (i) runs in polynomial time, (ii) gives a good approximate solution, *and* (iii) can be derandomized.

   (a) Consider an algorithm RANDMAXKSAT that takes as input an instance of MAX kSAT with n variables and m clauses, and assigns a value 0 or 1 to each variable uniformly and independently at random. Thus each variable has the probability exactly $1/2$ of getting the value 1.

      i. What is the probability that a given clause is satisfied by this algorithm?

      ii. What is the expected number of clauses satisfied by this algorithm?

      iii. Conclude that for *any* instance F of MAX kSAT with n variables and m clauses, there *exists* an assignment that satisfies at least $f(k) \cdot m$ clauses, for some function

[1] If you don't know what any of these terms mean, look it up, e.g., on Wikipedia.

$f(k)$ which is independent of n and m, and which approaches 1 as k grows larger. What is the form of $f(k)$ that you get?

iv. What is the probability that a given run of this algorithm will return an assignment that satisfies *at least* $f(k) \cdot m$ clauses?

(b) Derandomize the algorithm RandMaxkSAT by using the method of conditional expectations. Explain why your derandomized version is *guaranteed* to return an assignment that satisfies at least $f(k) \cdot m$ clauses.