1. One ordered pair $v = (v_1, v_2)$ dominates another ordered pair $u = (u_1, u_2)$ if $v_1 \geq u_1$ and $v_2 \geq u_2$. Given a set S of ordered pairs, an ordered pair $u \in S$ is called *Pareto optimal* for S if there is no $v \in S$ such that v dominates u. Give an efficient algorithm that takes as input a list of n ordered pairs and outputs the subset of all Pareto-optimal pairs in S. Provide a proof of correctness along with the runtime analysis.

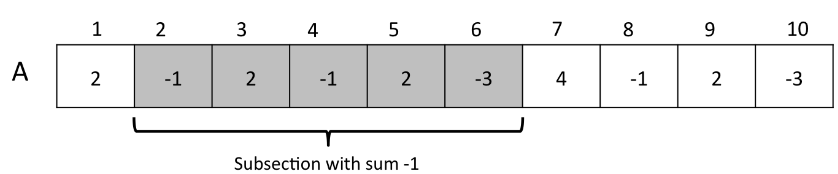
Solution:

Algorithm Description: Given an input of $(x_1, y_1), \dots, (x_n, y_n)$, if $n = 1$, return the single ordered pair (x_1, y_1), otherwise sort the ordered pairs by their x values. Use the y values as a secondary key to break ties in x values. Let $m = \lfloor n/2 \rfloor$ and split the input into $L = (x_1, y_1), \dots, (x_m, y_m)$ and $U = (x_{m+1}, y_{m+1}), \dots, (x_n, y_n)$. Then recursively find PL, PU, the pareto max subset of L, U, recursively. Then let yU be the maximum y value of U and let PLy be all the ordered pairs in PL that have a larger y value than yU. Then return $PLy \cup PU$.

Correctness: The base case works. Since all x values of L are lower than all x values in U, this means that there are no ordered pairs in L that dominate any ordered pair in U so all ordered pairs in the pareto max subset of U, PU must also be in the pareto max subset of the original input. Each ordered pair in PL has a lower x value than all ordered pairs in U so in order for an ordered pair in PL to be in the pareto max of the original set, it must have a higher y value than all ordered pairs of U. So, PLy is the set of all ordered pairs in PL that have a larger y value than all the ordered pairs in U.

Runtime: There is the cost of sorting. But this can be done as a preprocessing step. Then in the algorithm there are 2 recursive calls each of size $n/2$ and the non-recursive part of finding the max y value of U and finding all ordered pairs in PL that have a larger y value than the largest y value of U all can be done in $O(n)$. So this recursion has $a = 2, b = 2, d = 1$ and the algorithm will take $O(n \log(n))$.

2. Given a sequence of integers (positive or negative) in an array $A[1...n]$, the goal is to find a *subsection* of this array such that the sum of integers in the subsection is maximized. A subsection is a contiguous sequence of indices in the array. (*For example, consider the array and one of its subsection below. The sum of integers in this subsection is* -1.)



Let us call a subsection that maximizes the sum of integers, a *maximum subsection.* Design a divide and conquer algorithm with $O(n \log n)$ running time to output the sum of integers in a maximum subsection of a given array A. Give pseudocode and discuss running time.