Homework 11: Greedy cakes

You have a set of n cakes available, each with a certain satisfaction value si and an expiration date ei

(meaning it must be consumed at most in ei days starting from today). You can only eat one cake per day

(or will you get sick) and you must decide the best strategy to eat to maximize the total

satisfaction (sum of the values of the cakes you end up eating). This means you must decide which cake

you should eat in each day, starting in day 1 (remember, you can only eat one cake per day, at most).

To give an example, imagine you have the following set of n = 9 cakes:

i 1 2 3 4 5 6 7 8 9

si 10 20 10 15 14 40 18 5 20

ei 5 3 3 3 5 4 5 7 5

Because the maximum expiration date is day 7, we need to find which cake to eat in days 1 . . . 7. Cake i

can only be eaten in day j if j ≤ ei

, because of the expiration date. Suppose we indicate a possible allocation

of cakes to days with {c1, . . . , cn}, where cj is the index of the cake we eat in day j (with ” − ” indicating

that no cake is eaten on that day). Here are some possible valid allocations of cakes to days:

• {1, 2, 3, 5, 7, 8, −} which would have total satisfaction 77 = 10 + 20 + 10 + 14 + 18 + 5

• {1, 2, 3, 6, 7, −, 8} which would have total satisfaction 103 = 10 + 20 + 10 + 40 + 18 + 5

• {2, 4, 6, 7, 9, −, 8} which would have total satisfaction 118 = 20 + 15 + 40 + 18 + 20 + 5

For this particular instance, 118 is indeed the optimal total satisfaction, and no other allocation would

give a better value. And for a general case, can you devise an optimal strategy to follow?

1. Explain a greedy strategy that will give origin to an optimal solution, that is, one that maximizes

the total satisfaction.

Hint: you want to maximize satisfaction, so your strategy should take that into account.

2. Prove that the strategy you gave is optimal.

Hint: you must show that no other solution would be better. You can, for instance, use the ”exchange

”argument, or the ”stay-ahead” strategy.

3. Explain an implementation of your greedy strategy that runs in O(n log n) time.

Hint: what operations do you need? what algorithms and/or data structures would give you the needed

execution time guarantees?