

CSE 6140 - Homework 1

Alexander Winkles

Problem 1

- (a) Let $k \geq \lceil \log n \rceil$. We wish to design an algorithm of complexity $O(\log n)$. This suggests the idea of an algorithm similar to the methodology behind the Bisection Method.

Algorithm 1 Bisection Search for Box Dropping

```
1: procedure BISECTIONSEARCH( $l, n$ )
2:    $j \leftarrow n$                                      ▷ Highest Floor
3:    $i \leftarrow l$                                      ▷ Lowest Floor
4:    $m \leftarrow \frac{i+j}{2}$ 
5:   if  $m = \text{break}$  then                               ▷ Checks if box breaks on floor m
6:     BISECTIONSEARCH( $1, m - 1$ )
7:   else
8:     BISECTIONSEARCH( $m, n$ )
   return  $m$ 
```

The nature of the algorithm guarantees that this algorithm will take at most $\log n$ steps, so we will not run out of boxes.

- (b) Let $k < \lceil \log n \rceil$. We wish to design an algorithm of complexity $O(k + \frac{n}{2^{k-1}})$.

Algorithm 2 Box Dropping for $k < \lceil \log n \rceil$

```
1: procedure SOLVE( $n, k$ )
2:   BINARYSEARCH( $1, n$ )
3: procedure BINARYSEARCH( $l, n$ )
4:    $j \leftarrow n$                                      ▷ Highest Floor
5:    $i \leftarrow l$                                      ▷ Lowest Floor
6:    $m \leftarrow \frac{i+j}{2}$ 
7:   if  $m = \text{break}$  then                               ▷ Checks if box breaks on floor m
8:     BINARYSEARCH( $1, m - 1$ )
9:   else
10:    BINARYSEARCH( $m, n$ )
   return  $m$ 
```

- (c) Let $k = 2$. We wish to design an algorithm of complexity $O(\sqrt{n})$.

Algorithm 3 Box Dropping for $k = 2$

```
1: procedure FLOORSEARCH( $n$ )
2:    $m \leftarrow 0$  ▷ Initializes maximum floor
3:    $Nstep \leftarrow \sqrt{n}$  ▷ Number for floors to iterate over
4:   for  $i = 1 : n : Nstep$  do
5:
6:     if  $i \neq \text{break}$  then
7:        $m \leftarrow i$ 
8:       break
9:   for  $i = m + 1 : m + \sqrt{n}$  do
10:
11:    if  $i \neq \text{break}$  then
12:       $m \leftarrow i$ 
13:  return  $m$ 
```

Problem 2

Algorithm 4 Greedy 1

```
1: procedure MINIMIZESTRIPS( $C, n$ )
2:    $m \leftarrow 1$  ▷ number of strips needed
3:   Sort nodes  $c_i$  from leftmost to rightmost so that  $c_1 < c_2 < \dots < c_n$ 
4:    $minNode \leftarrow c_1$ 
5:   for  $i = 1 : n$  do
6:     if  $c_i - minNode > 9$  then
7:        $m \leftarrow m + 1$ 
8:        $minNode \leftarrow c_i$ 
9:   return  $m$ 
```

Essentially, this algorithm works by sorting all the nodes (“holes”) from left to right. From here, it starts at the leftmost strip then adds other nodes that will fit within the 9 inches of strip. Once all the nodes for a strip are filled, a new strip is added to the next node and the process is repeated. Given that this has a single for loop, its time complexity is $T(n) \in O(n)$.

Consider the subproblem of several nodes clustered together in such a way that there is no more than 9 inches between the leftmost and rightmost nodes. With our greedy choice, this subproblem will be covered completely by a single strip, which is trivially optimal. Thus, as the algorithm will treat all such subproblems this way, we can inductively see that the algorithm is optimal.

Problem 3

Algorithm 5 Greedy 2

```
1: procedure MAXIMIZEVALUE( $L, M$ )
2:   weightUsed  $\leftarrow 0$ 
3:    $B \leftarrow \emptyset$ 
4:   sort minerals  $m_i$  by  $\frac{v_i}{w_i}$  from highest value to lowest
5:   for  $m_i \in M$  do
6:     if weightUsed +  $w_i \leq L$  then
7:        $B = B \cup \{m_i\}$ 
8:       weightUsed +  $w_i$ 
9:        $M = M - \{m_i\}$ 
10:    break
11:  if weightUsed <  $L$  then
12:    pick the mineral next in line in  $M$  and call it  $m$ , with weight of  $w$  and value of  $v$ 
13:     $B = B \cup \{m * \frac{w}{L - \text{weightUsed}}\}$ 
14:  return  $B$ 
```

In order for this algorithm to be optimal, the bag must be completely filled in order to maximize profits.

Theorem 1. *The bag has been completely filled by the algorithm.*

Proof. This result is easy to see. At the end of the for loop, there are two possibilities:

(a) weightUsed = L

(b) weightUsed < L

If it is the first case, we are done. However, if it is the second case, then we can fit $L - \text{weightUsed}$ amount into the bag. Our algorithm provides this by adding a portion of the next mineral scaled with weight $L - \text{weightUsed}$, so after this our bag is again full. This completes our proof. \square

Theorem 2. *The greedy algorithm is optimal.*

Proof. Consider an optimal algorithm \mathcal{O} . Our goal is to use an exchange argument to show that the greedy solution A is optimal. Arrange the minerals in both as such in the algorithm, so m_i is first if and only if v_i/w_i is largest. Consider every mineral o_i in \mathcal{O} that differs from a_i in A . We can exchange this mineral with the corresponding mineral in A . If they are of different weights, we can instead exchange it for the appropriate portion of a_i . By definition of the greedy algorithm, $v_{a_i} \geq v_{o_i}$, where the equality holds otherwise \mathcal{O} is not optimal. By continuously applying the above procedure, we arrive at an optimal solution identical to A , so thus A must be optimal. \square

Problem 4

For this problem, I chose to implement Prim's algorithm in Python. I chose this problem because it was the less intuitive of the two for me, so I thought that writing code for it would help me gain a better understanding. Rather

than designing a second algorithm for `recomputeMST`, I chose to instead append the new edge to my graph then run `computeMST` again.