Problem Set 3

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Part A questions are due Tuesday, October 20th at 11:59PM.
Part B questions are due Thursday, October 22nd at 11:59PM.

Solutions should be turned in through the course website in PDF form using \LaTeX{} or scanned handwritten solutions.
A template for writing up solutions in \LaTeX{} is available on the course website.
Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, October 20th

1. (25 points) \(d\)-ary Heaps

In class, we’ve seen binary heaps, where each node has at most two children. A \(d\)-ary heap is a heap in which each non-leaf node (except perhaps one) has exactly \(d\) children. For example, this is a 3-ary heap:

(a) (2 points) Suppose that we implement a \(d\)-ary heap using an array \(A\), similarly to how we implement binary heaps. That is, the root is contained in \(A[0]\), its children are contained in \(A[1 \ldots d]\), and so on. How do we implement the \textsc{Parent}(\(i\)) function, which computes the index of the parent of the \(i\)th node, for a \(d\)-ary heap?

(b) (2 points) Now that there might be more than two children, \textsc{Left} and \textsc{Right} are no longer sufficient. How do we implement the \textsc{Child}(\(i, k\)) function, which computes the index of the \(k\)th child of the \(i\)th node? \((0 \leq k < d)\)
(c) **(5 points)** Express, in asymptotic notation, the height of a $d$-ary heap containing $n$ elements in terms of $n$ and $d$.

(d) **(5 points)** Give the asymptotic running times (in terms of $n$ and $d$) of the `HEAPIFY` and `INCREASE-KEY` operations for a $d$-ary heap containing $n$ elements.

(e) **(6 points)** Let’s suppose that when we build our $d$-ary heap, we choose $d$ based on the size of the input array, $n$. What is the height of the resulting heap (in terms of $n$) if we choose $d = \Theta(1)$? What if $d = \Theta(\log n)$? What about $d = \Theta(n)$? (HINT: remember that $\log_d n = \frac{\log n}{\log d}$. This might simplify your expressions a little.)

(f) **(5 points)** What are the running times of `HEAPIFY` and `INCREASE-KEY` for the three choices of $d$ above? Do the running times increase or decrease when you increase $d$? If your program calls `HEAPIFY` and `INCREASE-KEY` the same number of times, what would be your choice for $d$ and why?

2. **(25 points)** Visible Trees

There are $n$ trees in a street, each given with its location $x_i > 0$ and height $h_i > 0$ ($i = 1, \ldots, n$). Trees are given in arbitrary order of their location. Professor Devadas is standing at location $x_0 = 0$ and wondering how many trees he can see from there. Please, help him compute this number. A tree is visible if any part of it is visible. Assume that all trees are much taller than Professor Devadas; i.e., $h_0 = 0$. For example, dark colored trees are visible in this picture:

(a) **(5 points)** Write function `blocks(x1, h1, x2, h2)` that returns `True` if tree $(x_1, h_1)$ blocks Professor’s view of tree $(x_2, h_2)$ and `False` otherwise.

(b) **(3 points)** A 6.006 alumnus suggests the following algorithm for computing the number of visible trees ($x[i]$ and $h[i]$ correspond to $x_i$ and $h_i$):
def count_visible(x, h):
    n_visible = 0
    for i in range(1,n+1):
        i_is_blocked = False
        j = 1
        while j <= n and not i_is_blocked:
            if j != i and blocks(x[j],h[j],x[i],h[i]):
                i_is_blocked = True
            j = j + 1
        if not i_is_blocked:
            n_visible = n_visible + 1
    return n_visible

What is the asymptotic time complexity of this algorithm?

(c) (5 points) Give one example of a worst case and one example of a best case
scenario for the previous algorithm for \( n = 5 \). How many times the function
blocks() is called in each case?

(d) (12 points) Professor Devadas knows you can do better. Describe an \( O(n \log n) \)
solution to the problem. Explain why your algorithm runs in \( O(n \log n) \) time.
Note: You do not need to write a working python code. However, you can
write a python style description of your algorithm, similar to count_visible() above. You can use any algorithm given in lecture without implementing it.
Alternatively, you can describe your solution in plain words.

Part B: Due Thursday, October 22nd

(50 points) Pset Scheduling

Ben Bitdiddle is behind on his problem sets. In fact, he is already late on \( N \) different prob-
lem sets (\( 1 \leq N \leq 100.000 \)). Fortunately for Ben, all of his classes accept late homework
with a grade penalty for each day late.

Suppose that problem set \( i \), where \( i \) is in \{1 \ldots N\}, takes \( D_i \) days to complete and has a
penalty of \( P_i \) points per day late. There is no limit to the number of penalty points Ben
can accrue. (Ben’s penalty adjusted score can become negative.) Ben is required to finish
each problem set.

Help Ben by writing a program to determine the order in which Ben should do his prob-
lem sets in order to minimize the total number of penalty points Ben receives on all of
his assignments. Your program should read from standard input and write to standard
output.
As part of your program, you will need to do some sorting. You should write your own
implementation of heap sort for this problem. Your implementation of heap sort should
have the signature:

```python
def heap_sort(list):
```

where `list` is the list to sort. Your function should sort `list` in ascending order using
the default Python ordering defined by `<` and `>`. Using this function specification will
allow us to better test your code if you have a bug and give you more partial credit.

**Input Format**

**Line 1:** The single integer $N$.

**Lines 2...$(N + 1)$:** The $(i + 1)$-st line describes the Ben’s $i$-th problem set and con-
tains two integers, $D_i$ and $P_i$, separated with a single space.

**Sample Input**

```
4
4 1
2 5
1 2
2 3
```

**Output Format**

**Line 1:** A single integer which is the minimum possible number of penalty points
Ben can receive.

**Sample Output**

```
40
```

**Sample Explanation**

In the sample above, Ben first spends 2 days finishing pset #2, accruing 10 penalty
points. He then spends 1 day finishing pset #3, accruing 6 penalty points. He then 2
days finishing pset #4, accruing 15 penalty points. He lastly spends 4 days finishing
pset #1, accruing 9 penalty points.

**Hint:** Think about how you might approach this in real life. You should somehow prior-
itize Ben’s problem sets and then sort them according to this priority.