

Luke Erlewein

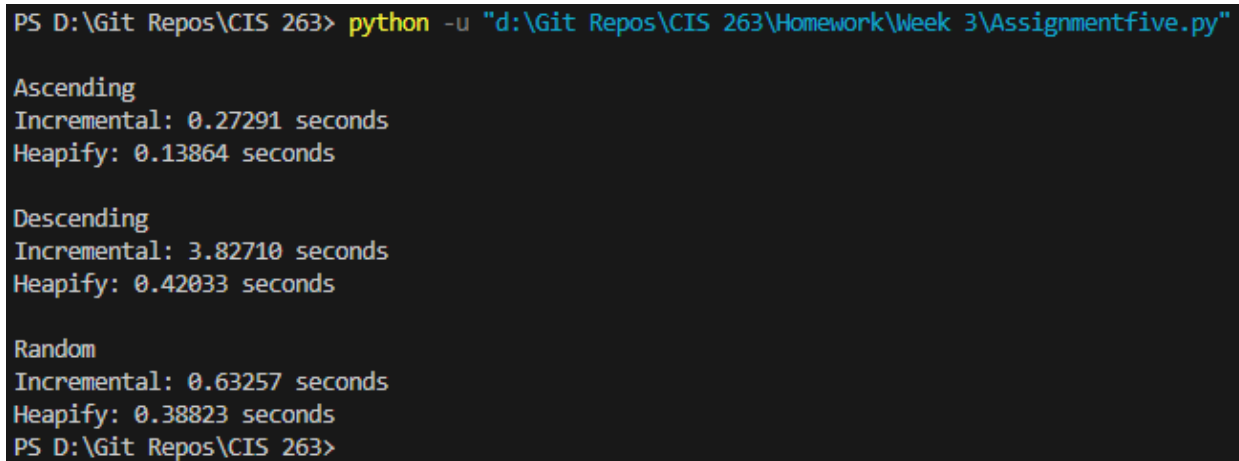
Dr. Denton Bobeldyk

CIS 263

7/12/2025

Assignment 5

Program Output:



```
PS D:\Git Repos\CIS 263> python -u "d:\Git Repos\CIS 263\Homework\Week 3\Assignmentfive.py"

Ascending
Incremental: 0.27291 seconds
Heapify: 0.13864 seconds

Descending
Incremental: 3.82710 seconds
Heapify: 0.42033 seconds

Random
Incremental: 0.63257 seconds
Heapify: 0.38823 seconds
PS D:\Git Repos\CIS 263>
```

Figure 1: Output of the program.

Output Analysis:

With an N value of 1,000,000, we are able to get meaningful time differences. It is expected that the descending method will be the slowest of the methods, as each new number that is being analyzed is a smaller number. This means that there are more switches happening because every single new number that is read in will cause a comparison and switch. The ascending is already in the correct order of smallest to largest, so it is merely just a comparison for that one, which leads to the ascending to be the one that is the fastest option. The random method is between the two other methods, which could be because there is

As for the comparison between the incremental insertion and the heapify method, from the data presented, it seems as if the heapify method is the faster one. The reason for this is that

the insertion builds the heap one node at a time and then adjusts the nodes above that one as needed. However, the heapify method looks at the already built entire node structure and then looks at the last parent node and fixes it up to the root. This covers all the nodes faster as it leaves entire branches checked. Therefore, you do not need to cover branches that you have already looked over.

Number 4 Solution:

First, let's define what a d-heap is. It is a heap structure where each node has d children. The kind of heap we made in our original program was a 2-heap. Or a binary heap where each node only has two children.

For this, we are going to have the first element in the array be at 1, meaning that the index starts at 1. In order to find out where in the array the children of a node is we can look to the following formula.

$$child_k(i) = d \cdot (i - 1) + k + 1 \text{ for } k = 1 \text{ to } d \quad (1)$$

Equation 1 gives the formula for all d children in the array.

So, for an example of how this works, let's assume we have an array with a d value of 3, meaning that each node has three children. Our array will be the following.

[A, B, C, D, E, F, G, H, I, J]

There are 10 different values in the array, but then we can figure out what the children of each of the nodes are by using our formula. If we try to find the children of node A, which is index 1, we can plug 1 into the formula, and we get

$$child_k(i) = 3 \cdot (1 - 1) + 1 + 1 = 2$$

$$child_k(i) = 3 \cdot (1 - 1) + 2 + 1 = 3$$

$$child_k(i) = 3 \cdot (1 - 1) + 3 + 1 = 4$$

So that means that the children are indexes 2, 3, and 4. If we want to now find the children of B, which is index two, we can now plug that into our formula and get the following.

$$child_k(i) = 3 \cdot (2 - 1) + 1 + 1 = 5$$

$$child_k(i) = 3 \cdot (2 - 1) + 2 + 1 = 6$$

$$child_k(i) = 3 \cdot (2 - 1) + 3 + 1 = 7$$

So the children for node B are indexes 5, 6, and 7. This can be done for any node in order to find the children of that node.

$$parent(i) = \frac{i-2}{d} + 1 \quad (2)$$

Equation 2 gives the formula for the parent node of node i.

In order to show the proof that this equation is the correct one, let's find the parent node of node

F. Node F is in index 6. So the i in this case will be 6.

$$parent(i) = \frac{6-2}{3} + 1 = 2.3333$$

If you round down, you get two. And two is the index of B. We can determine this to be the correct answer from the calculations we did to find the child nodes of B, where F was one of the child nodes.