CS 310
Assignment 207

William Briggs

Problem 2. Analyze your two methods. Explain what operations you are counting, what the input size is, and what your results are.

```
1   void reverse()
2   {
3     Node * node_ptr {first};    //keeps track of the current node
4     Node * prev_ptr {nullptr};  //keeps track of the previous node
5     Node * next_ptr {nullptr};  //keeps track of the next node
6     while(node_ptr != nullptr)
7     {
8       next_ptr = node_ptr->next; //moves the next pointer down the list
9       node_ptr->next = prev_ptr; //current node points to prev node
10      prev_ptr = node_ptr;       //shifts prev pointer to current pointer
11      node_ptr = next_ptr;       //shifts current pointer to the next pointer
12    }
13    //sets the head of the list to the last node
14    first = prev_ptr;
15  }
```

For the reverse() method, I count every assignment and comparison as a basic operation. Looking at lines 3-5, 14 these total a static 4 operations. This brings us to the while statement, this statement should run n amount of times, where n is the size of the singly linked-list. Within this statement there are 4 assignments (lines 8-11). So in total I found this method to run in:

$$t(n) = 4n + 4$$

$$t(n) \in \Theta(n)$$

```
1   void push_back (const Object& item)
2   {
3     auto new_node {new Node (item)};
4     if (empty())
5     {
6       first = new_node;
7     }
8     else
9     {
10      auto node_ptr {first};
11
12      //iterates through the list until a node that points to null is found
13      while(node_ptr->next != nullptr)
14      {
15        node_ptr = node_ptr->next;
16      }
17      //adds the new node to the end of the list
18      node_ptr->next = new_node;
```

```
19        }
20        size++;
21    }
```

For this method, I also am counting every assignment and comparison as a basic operation. At lines 3, 10, 18, and 20 we have a static 4 operations for the worst-case. With the while statement running n times, where n is the size of the singly linked-list, with one assignment inside that loop. I found this method to run in:

$$t(n) = n + 4$$

$$t(n) \in \Theta(n)$$

I found the algorithm analysis for both of these methods to be $\Theta(n)$ as there is no possible variations in operation count, given the static input size.

Problem 3. Assuming the data in a heap are all distinct, what are the possible locations of: Explain and justify your answers

The smallest element: It must be located at the lowest or second lowest tier as it is a terminal node, having no child nodes. In terms of location in the array, it will be after the $\lceil \frac{n}{2} \rceil^{th}$ index of the array. This is because the max heap has the property defining trait of each parent node having to be larger than the child nodes.

The second-smallest element: This will either be a sibling, parent, or cousin to the smallest element. In terms of location in the array, it will either be the $\lceil \frac{n}{2} \rceil^{th}$ index of the array or come after it. This is because it is possible for the second smallest element to be the parent of the smallest element, as it is by definition greater than the smallest element.

The second-largest element: Will be a child node of the largest element, as the largest element is the root. Location in an array will either be index(1) or index(2), given that the root is index(0). This is because there is only one value greater than it and the second largest element must be a child of the root node; and because the child nodes of a binary heap are not ordered, there are two possibly locations for it in an array.