CS 310

Assignment 0911 - Algorithm Analysis

Bill Jin

Analysis for push function:

1.**push** function code :

```
1  void push(unsigned element)
2  {
3    heap.push_back(element);
4    bubble_up(heap.size() - 1);
5  }
```

2.**bubble_up** function code :

```
1  void bubble_up(size_t position)
2  {
3    if (position != 0)
4    {
5      size_t parent = (position - 1) / 2;
6      while (position != 0 && heap.at(position) > heap.at(parent))
7      {
8        std::swap(heap.at(position), heap.at(parent));
9        position = parent;
10       parent = (position - 1) / 2;
11     }
12   }
13 }
```

Analysis: This algorithm is used to place the node into its right place by using while loop swapping its parent node with if the node is greater than that. And for the input size will be the size of the heap we build.

For the **worst cases** (Big-O), the operations that are counted are:

In push function :

- the push_back function, count as 1 operation,

In bubble_up function :

- the if condition comparison on line 3, count as 1 operation,
- the assignment of parent, count as 1 operation,
- while loop condition, since it has 3 comparisons, count as $\lfloor \log_2(n+1) \rfloor \times 3$ operations,
- swap function, count as $\lfloor \log_2(n+1) \rfloor \times 2$ operations,
- assignment of position, count as $\lfloor \log_2(n+1) \rfloor \times 1$ operations,
- parent assignment on line 10, count as $\lfloor \log_2(n+1) \rfloor \times 1$ operations,

- one more for loop condition check, count as $\lfloor \log_2(n+1) \rfloor \times 3$ operations.

After sum up all, the number of times of all operations that are executed is:

$$\lfloor \log_2(n+1) \rfloor \times 10 + 3$$

For the **best cases** (Big-Ω), the operations that are counted are:

In push function :

- the push_back function, count as 1 operation.

In bubble_up function :

- we assume it only has root which means only run if condition, count as 1 operation.

After sum up all, the number of times of all operations that are executed is :

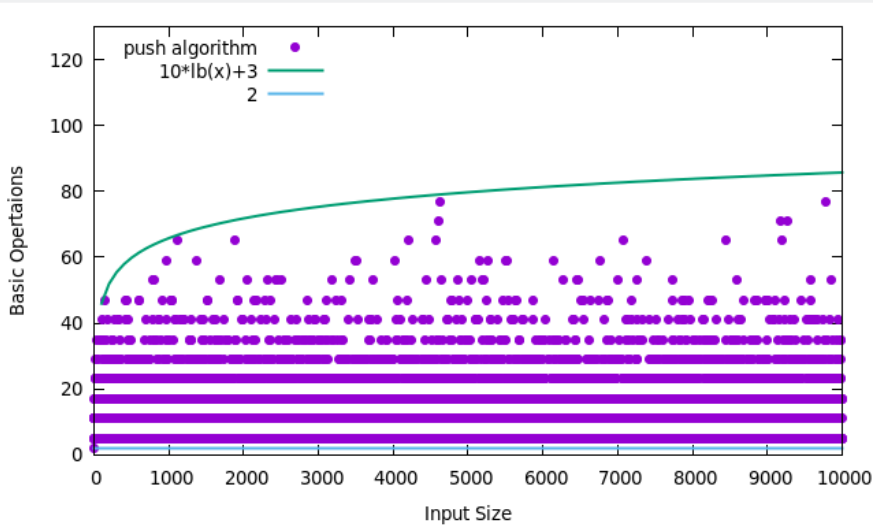
2 operations

Since there are while loop and if conditions which could cause the best and the worst cases. Therefore, we can find out that this algorithm belongs to Big-Oh and Big-Omega efficiency class.

$$T(n) \in O(\log_2(n)) \text{ for } c = 10, \text{ all } n \geq 0$$

$$T(n) \in \Omega(1)$$

And the resulting data file is plotted, we get the following. Also plotted on the same axes are the scaled standard functions $\log_2(n+1) \times 10 + 3$ and 2 which illustrate that the algorithm belongs to big-O and big-Ω .



Analysis for pop function:

1.**pop** function code :

```
unsigned pop()
{
  assert(heap.size() > 0);
  unsigned value_to_return = heap.at(0);
  heap.at(0) = heap.at(heap.size() - 1);
  heap.pop_back();
  percolate_down(0);
  return value_to_return;
}
```

2.**percolate_down** function code :

```
void percolate_down(size_t position)
{
  size_t left_child_position;
  size_t right_child_position;

  left_child_position = 2 * position + 1;
  right_child_position = 2 * position + 2;

  bool loop_guard = true;

  while (number_of_chidren(position) >= 1 && loop_guard)
  {
    if (number_of_chidren(position) == 2)
    {
      if (heap.at(left_child_position) <= heap.at(right_child_position))
      {
        if (heap.at(position) <= heap.at(right_child_position))
        {
          std::swap(heap.at(position), heap.at(right_child_position));
          position = right_child_position;
        }
        else
        {
          loop_guard = false;
        }
      }
      else
      {
        if (heap.at(position) <= heap.at(left_child_position))
        {
          std::swap(heap.at(position), heap.at(left_child_position));
          position = left_child_position;
        }
```

```
34          else
35          {
36            loop_guard = false;
37          }
38        }
39      }
40      else if (number_of_chidren(position) == 1)
41      {
42        if (heap.at(position) < heap.at(left_child_position))
43        {
44          std::swap(heap.at(position), heap.at(left_child_position));
45          position = left_child_position;
46        }
47        else
48        {
49          loop_guard = false;
50        }
51      }
52      left_child_position = 2 * position + 1;
53      right_child_position = 2 * position + 2;
54    }
55    ((void)0);
56  }
```

Analysis: This algorithm is used to put the node into its right place by using while loop swapping its children node with. And for the input size will be the size of the heap we build.

For the **worst cases** (Big-O), the operations that are counted are:

1. **pop function** :
   - the assert function, count as 1 operation,
   - value_to_return assignment count as 1 operation,
   - assignment of root node, count as 1 operation,
   - heap.pop function, count as 1 operation,
   - return value, count as 1 operation.
2. **percolate_down function** :
   - two children declarations and assignment from line 3 to 7, count as 4 operations,
   - loop_guard declaration assignment, count as 1 operation,
   - while condition, count as $\lfloor \log_2(n+1) \rfloor \times 2$ operations,
   - if condition for worst case, count for $\lfloor \log_2(n+1) \rfloor \times 1$ operation,
   - next if condition on line 17, count for $\lfloor \log_2(n+1) \rfloor \times 1$ operation,
   - swap function on line 19, count as $\lfloor \log_2(n+1) \rfloor \times 2$ operations,
   - else if condition check, count as $\lfloor \log_2(n+1) \rfloor \times 1$ operation,

- both left and right side children assignments, count as $\lfloor \log_2(n+1) \rfloor \times 2$ operations
- while loop count one more time, count as $\lfloor \log_2(n+1) \rfloor \times 2$ operations

After sum up all, the number of times of all operations that are executed is:

$$\lfloor \log_2(n+1) \rfloor \times 11 + 10$$

For the **best cases** (Big-Ω), the operations that are counted are:

1. **pop function** :
   - the assert function, count as 1 operation,
   - value_to_return assignment count as 1 operation,
   - assignment of root node, count as 1 operation,
   - heap.pop function, count as 1 operation,
   - return value, count as 1 operation.
2. **percolate_down function** :
   - two children declarations and assignment from line 3 to 7, count as 4 operations,
   - loop_guard declaration assignment, count as 1 operation,
   - heap only has one node, the root, so while loop will not run, count only the condition 2 operations.

After sum up all, the number of times of all operations that are executed is:

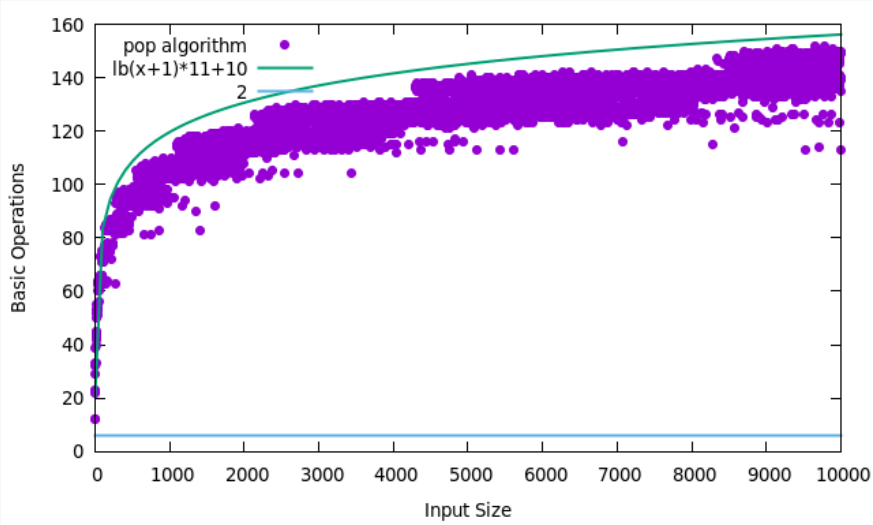
$$6 \text{ operations}$$

Since there are while loop and if conditions which could cause the best and the worst cases. Therefore, we can find out that this algorithm belongs to Big-Oh and Big-Omega efficiency class.

$$T(n) \in O(\log_2(n)) \text{ for c} = 11, \text{ all n} \geq 1$$

$$T(n) \in \Omega(1)$$

And the resulting data file is plotted, we get the following. Also plotted on the same axes are the scaled standard functions $\log_2(n+1) \times 11 + 10$ and 6 which illustrate that the algorithm belongs to big-O and big-Ω .


pop algorithm
lb(x+1)*11+10
2
Basic Operations
Input Size
0
20
40
60
80
100
120
140
160
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000