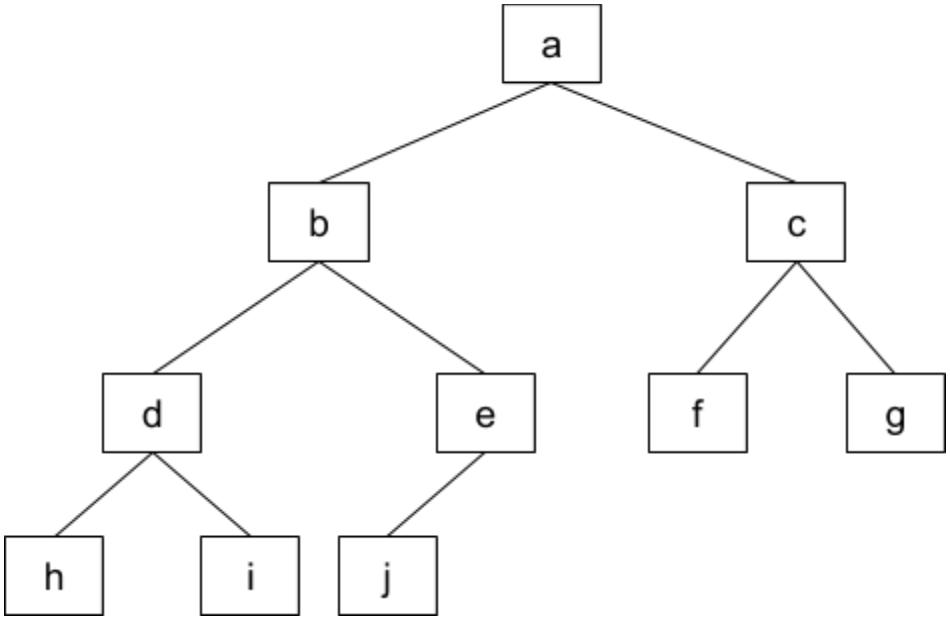
Jessica Spokoyny
Assignment 5 Extra Credit
May 18, 2016

Assignment 5 - Extra Credit

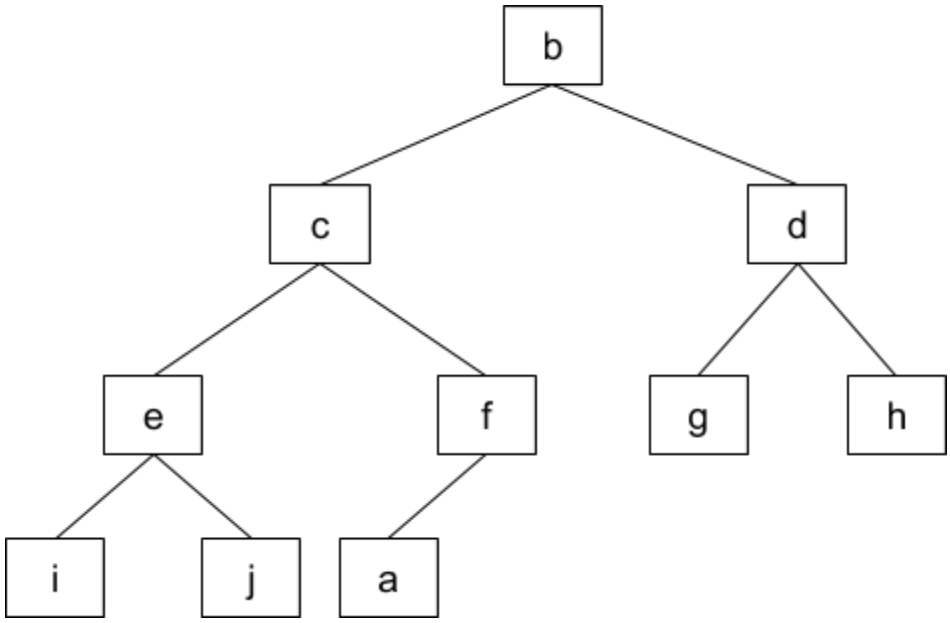
If we were to add 10 tasks with the same priority to the priority queue:

- Task 1: [name: a , priority: 5]
- Task 2: [name: b , priority: 5]
- Task 3: [name: c , priority: 5]
- Task 4: [name: d , priority: 5]
- Task 5: [name: e , priority: 5]
- Task 6: [name: f , priority: 5]
- Task 7: [name: g , priority: 5]
- Task 8: [name: h , priority: 5]
- Task 9: [name: i , priority: 5]
- Task 10: [name: j , priority: 5]

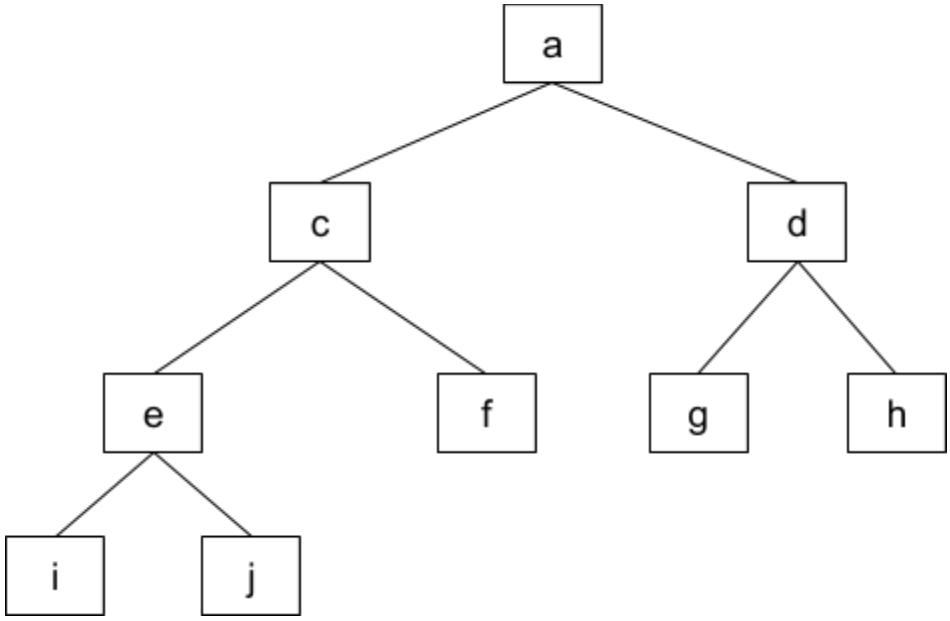
The resulting queue would look as follows:



After we call sort, it becomes:



Then, we remove the min and get:



If we go through and sort/remove min procedures 10 times, the order of removal is:

- b
- c
- d
- e
- f
- g
- h
- i
- j
- a

This is not what we would expect. We would want the sort and remove min functions to account for the scenario where the priorities are equal and remove those elements in the order in which they were added to the list.

To modify the functions to account for this scenario, I added a conditional check before swapping. If the priorities of the first and last elements are equal, they need not be swapped (as it must be true that all other elements between the first and last must necessarily have that same priority as well). The changes can be seen in blue below:

```
void dyHeapRemoveMin(DynamicArray* heap, compareFunction compare)
{
   assert(heap != NULL);

   // last element of the heap
   int last = dySize(heap) - 1;

   assert(last >= 0);

   // if priorities of first and last elements are not equal
   if (!(compare(dyGet(heap, 0), dyGet(heap, dySize(heap) - 1)) == 0))
   {
      // assign the value of the last element to be the value of the first
      heap->data[0] = heap->data[last];

      // remove the last element
      dyRemoveAt(heap, last);
   }

   else
```

```
    {
        // remove the first element
        dyRemoveAt(heap, 0);
    }

    adjustHeap(heap, last, 0, compare);
}


void dyHeapSort(DynamicArray* heap, compareFunction compare)
{
    assert(heap != NULL);

    buildHeap(heap, compare);

    int i;
    for(i = dySize(heap) - 1; i > 0; i--)
    {
        // only swap if priorities are not equal
        if (!(compare(dyGet(heap, 0), dyGet(heap, dySize(heap) - 1)) == 0))
            dySwap(heap, 0, i);
        adjustHeap(heap, i, 0, compare);
    }
}
```