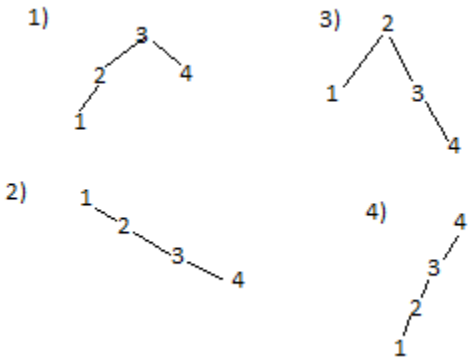
HW5 Theory

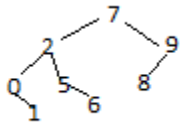
Alexander Kazantsev

February 10, 2016

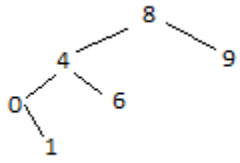
Problem 5.1



Problem 5.2



Problem 5.3



Problem 5.4

The order in which elements in a binary tree are deleted do no affect the final order of the binary tree.

Problem 5

n	T(n)
10	2.59876251221e-05
100	0.000167846679688
1000	0.00176882743835
10000	0.0182538032532
100000	0.186135053635
1000000	1.9062898159

Without the aide of a plot it is easily visible that $T(n)$ grows linearlly with the growth of n, $MH(n) \in \mathrm{O}(n)$

Problem 6

It has been established that to build a heap it takes approxmiately linear time, which means it is bounded by $\mathrm{O}(n)$

Next, the first and last element are swapped in the array and the size is decremented which is constant, O(1)

Finally, the first element is down heaped and the cycle continues from the second step. Each down heap costs $\mathrm{O}(log(n))$ and is repeated n times, which yields a time complexity of $\mathrm{O}(nlog(n))$ using $BigOh$ multiplicative rule. The size of the actual algorithm is constant as only two items are swapped at every iteration.

Problem 7

```
otherDS* holdOrder(otherDS data)
    if first(data) == NULL
        return NULL
    otherDS *result = malloc( sizeof(otherDS) * len(data))
    result[0] = *first(data)
    count = 1
    while next(data) != NULL
        result[count] = *next(data)
        count += 1
```

```
    return result
end
double *copyHeight (otherDS data)
    if first(data) == NULL
        return NULL
    float *result = malloc( sizeof(flaot) * len(data))
    result[0] = first(data)->height
    count = 1
    while next(data) != NULL
        result[count] = next(data)->height
        count += 1
    return result
end
void copyComputedValue( doubles d[], otherDS data)
    first(data)->computed value = d[0]
    count = 1
    temp = next(data)
    while temp != NULL
        temp->computed value = d[count]
end
main()
    otherDS data = holdOrder( satelliteData )
    float *allHeight = copyHeight( data )
    copyComputedValue( impressiveA( allHeight ), data)
```

All of the functions written above operate at O(n) as they only iterate over n elements. Since none of the operations are embedded the additivity rule can be used to determine the whole run time is at O(n). The space complexity is also O(n)

Problem 8

.

.

```
struct SQ( // stack queue
        stack A
        stack B
)
Enqueue (SQ q, element x)
        push(q.A, x)
end
Remove(SQ q)
        while q.A not empty
                push( q.B, pop( q.A ) )
        result = pop(q.B)
        while q.B not empty
                push( q.A, pop( q.B ) )
        return result
end
```

The time complexity for insert in this implementation is constant, and the time complexity for remove is linear. The space complexity is $2n$ which simplifies to $O(n)$

Problem 9

Math and all that jazz

I'm tired and I have to take the exam in 11 hours. So how about I tell you my favorite color. It's blue, becuase the sky is blue.