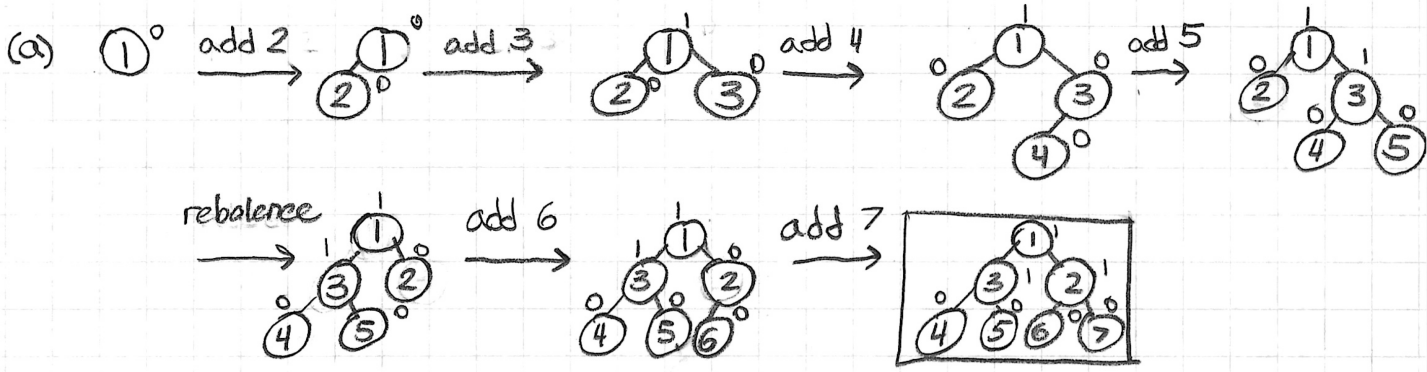
CS340 => Assignment 4

Maksim Sharoika

Problem 1)

(a)



(b) Lets say n is the size of our heap.

↳ at first we will enque and merge all single this would O(N/2) of this operation, the decieve is constant time while the enqueue will be logarithmic time, this would then be extend to half of the merged and so on.

1. $O(n/2) \rightarrow O\ log(2)$
2. $O(n/4) \rightarrow O\ log(4)$
3. $O(n/8) \rightarrow O\ log(8)$

} this pattern would continue to grow till the denominator becomes that of size of n.

This gives us a pattern of:

$$\lim_{i \to \infty} \frac{i}{2^i} = 0$$

$$O(n/2\ log(2)) + O(n/4\ log(4)) + \ldots + O(n/n\ log(n))$$

$$\rightarrow O\left(n \sum_{i=1}^{\log n} \frac{1}{2^i} (\log(2^i))\right) \Rightarrow O\left(n \sum_{i=1}^{\log n} \frac{i\ \log(2)}{2^i}\right) \Rightarrow O\left(n \sum_{i=1}^{\log n} \frac{i}{2^i}\right)$$

↳ Therefore that the runtime will converge towards a constant value as operations become faster.

↳ $\boxed{O(n)}$

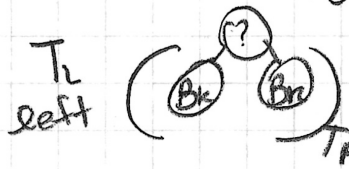
Problem 2)

We will have to prove that B_k always has children binomial trees $B_0, B_1 \ldots B_{k-1}$ by using inductive reason.

Base → $k = 1$, ∴ in any binomial tree where the root has only one child, this child must be B_0.

Hypothesis → Binomial tree B_k will always have binomial trees B_0, B_1, B_{k-1} as children of the root.

Induction → $k = k+1$, A binomial tree in B_{k+1} will have have one node taller of a root than B_k, then a binomial tree a size of B_k may exist in k arrant under it.



← keeps breeding T_{k+1}

Therefore the children of tree with B_{k+1} will have one B_k tree, and another B_k tree, this will continue from B_k to B_{k-1} to B_{k-2} all the way to B_2, B_1, and finally B_0. B_1 and B_0 being our base case.

T_k ← B_k tree

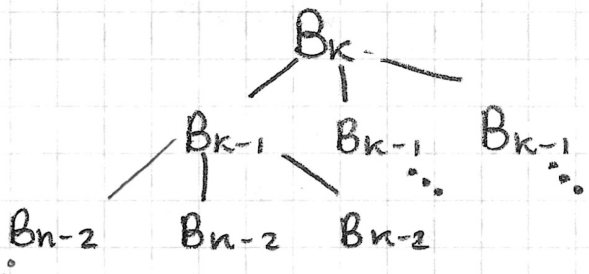
each time having k trees.

* Whereever we discuss using "tree size B_k" replace that with T_{B_k}.

Base

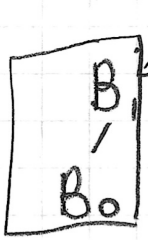
} Our hypothesis is that this always holds true.

Inductive Base



× # of k-1 times

× # of k-2 times.



} 2 times

} 1 time

as you can visually see this hold for a binomial tree.

Problem 3) $a = [1, \underset{m}{4}, 3, \underset{n}{2}, 5]$ } array example

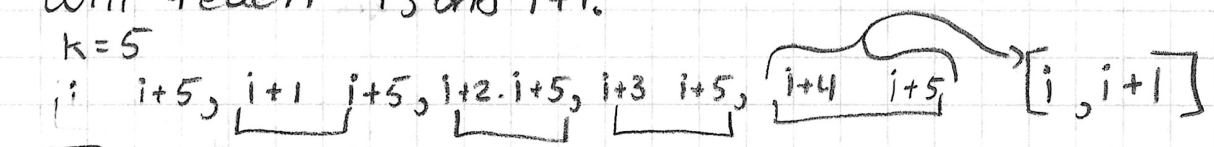
Since i and i+k are in the wrong order, and we flip them they must now be in the right order because they are numerical / comparable.
i.e. i->3 i+k->2 now i->2 and i+k->3 } fixed!

↳ Since we are using an array with an operation on the i and i+k locations the operation / inversion will only have those entries.

} The elements from i to 0 or from to i+k to max don't ever need to to be involved in a swap, thier position is not changed. Same for i+k to max.

Therefore when swap elements from outside in the wrong order from i to i+k the only elements that may be swaped are i+1, i+2... to i+k.
eventually the only possibilty for a swap will reach i, and i+1.

k=5



Therefore the set of i+1 to i+1 to i+k is equal to k-1, since each inversion requires 2 entries that may overlap we get 2k-2.

Adding the first inversion which would again use the k+1 indices we can simplify it to 2k-1.

Lower Bound (worst) 1 2 3 0 -> 0 2 3 1 -> 0 1 3 2 -> 0 1 2 3
(i, i+k), (i+1, i+k), (i+2, i+k) ... (i+k-1, i+k)
↳ we will have to perform many swaps if the array is in order but last element.

↳ On the other hand 1 0 1 2 0 will only requre one swap.
(i, i+k)

Problem 4)

(a) [1, 1, 1, 1, 1] → [1, 1, 1, 1, 1, 1] → this would then be O(N) because N is size.

This would be the best case for insertion sort because it would just need one traversal and then throw the correct value in the back.

(b) Insertion sort is a stable algorithm, we only swap values if the item on the right is less than the one on the left. Therefore the ordering of two equivalent entries should not swap.