Today’s announcements:

MP6 available, due 11/18, 11:59p.
Code challenge! 11/19, 9p, in Siebel 0224.
(min)Heap: removeMin
template <class T>
T Heap<T>::removeMin(){
    T minVal = items[1];
    items[1] = items[size];
    size--;
    heapifyDown(1);
    return minVal;
}

template <class T>
void Heap<T>::heapifyDown(int cIndex){
    if (hasAChild(cIndex)){
        minChildIndex = minChild(cIndex);
        if (items[cIndex] ____ items[minChildIndex]{
            swap(______________,______________);
            __________________________;
        }
    }
}
What have we done?
(min)Heap: buildHeap
template <class T>
void Heap<T>::buildHeap()
{
   for (int i=2;i<=size;i++)
      heapifyUp(i)
}

1. Sort the array:

2. template <class T>
void Heap<T>::buildHeap()
{
   for (int i=2;i<=size;i++)
      heapifyUp(i)
}

3. template <class T>
void Heap<T>::buildHeap()
{
   for (int i=parent(size);i>0;i--)
      heapifyDown(i)
}
Thm: The running time of buildHeap on an array of size $n$ is ________.

Instead of focussing specifically on running time, we observe that the time is proportional to the sum of the heights of all of the nodes, which we denote by $S(h)$.

$$S(h) = \quad S(0) =$$

Soln $S(h) =$

Proof of solution to the recurrence:

But running times are reported in terms of $n$, the number of nodes...
(min)Heap: heapSort

Running time?

Why do we need another sorting algorithm?
This image reminds us of a ____________
which is one way we can implement ADT ________________
whose functions include ________________ and ________________
whose running times are ____________________
This structure can be built in time ________________,
which helps us do a worst case time ____________ sort, in place.