BuildHeap and Heapsort
Goals

• Build a heap from an array of arbitrary values
• HeapSort algorithm
• Analysis of HeapSort
• How do we build a heap from an arbitrary array of values???
BuildHeap: Is this a proper heap?

Are any of the subtrees **guaranteed** to be proper heaps?
BuildHeap: Leaves are proper heaps

Size = 11
Size/2 – 1 = 4
• How can we use this information to build a heap from a random array?
• _adjustHeap: takes a binary tree, rooted at a node, where all subtrees of that node are proper heaps and percolates down from that node to ensure that it is a proper heap

```c
void _adjustHeap(struct DynArr *heap, int max, int pos)
```

- Adjust up to (not inclusive)
- Adjust from
BuildHeap

• Find the last non-leaf node, i, (going from left to right)
• adjust heap from it to max
• Decrement i and repeat until you process the root
• BuildHeap and _adjustHeap are the keys to an efficient, in-place, sorting algorithm
• in-place means that we don’t require any extra storage for the algorithm
• Any ideas???
HeapSort

1. BuildHeap – turn arbitrary array into a heap

2. Swap first and \textit{last} elements

3. Adjust Heap (from 0 to the \textit{last}...not inclusive!)

4. Repeat 2-3 but decrement \textit{last} each time through
HeapSort Simulation: BuildHeap

Max = 6
i = Max/2 – 1 = 2

i = 2
_adjustHeap(heap, 6, 2)
HeapSort Simulation: BuildHeap

Max = 6
i=1

i=1
_adjjustHeap(heap,6,1)
HeapSort Simulation: BuildHeap

Max = 6
i = 0

```python
_i=0
_adjustHeap(heap,6,0)
```
HeapSort Simulation: BuildHeap

i=-1
Done...with BuildHeap ....now let’s sort it!
1. BuildHeap

2. Swap first and last

3. Adjust Heap (from 0 to the last)

4. Repeat 2-3 but decrement last
HeapSort Simulation: Sort in Place

iteration 1

i=5
Swap(v, 0, i)
_adjustHeap(v, i, 0);
HeapSort Simulation: Sort in Place

Iteration 1

i = 5
Swap(v, 0, i)
adjjustHeap(v, i, 0);
HeapSort Simulation: Sort in Place

Iteration 2

\[ i = 4 \]

\[ \text{Swap}(v, 0, i) \]

\[ \text{_adjustHeap}(v, i, 0); \]

\[
\begin{array}{cccccc}
  0 & 1 & 2 & 3 & 4 & 5 \\
  4 & 5 & 7 & 9 & 8 & 3 \\
\end{array}
\]
i=4
Swap(v, 0, i)
_adjustHeap(v, i, 0);
HeapSort Simulation: Sort in Place

Iteration 3

```
i = 3
Swap(v, 0, i)
_adjustHeap(v, i, 0);
```
HeapSort Simulation: *Sort in Place*

```
i = 3
Swap(v, 0, i)
_adjustHeap(v, i, 0);
```
i = 2
Swap(v, 0, i)
_adjustHeap(v, i, 0);
HeapSort Simulation: Sort in Place

Iteration 4

i=2
Swap(v, 0, i)
_adjustHeap(v, i, 0);
i=1
Swap(v, 0, i)
_adjustHeap(v, i, 0);
HeapSort Simulation: Sort in Place

Iteration 5

0 1 2 3 4 5
9 8 7 5 4 3

\[ i = 1 \]
\[ \text{Swap}(v, 0, i) \]
\[ \_\text{adjustHeap}(v, i, 0); \]
HeapSort Simulation: Sort in Place

i=0
DONE
HeapSort Performance

• Build Heap:
  
n/2 calls to \_adjustHeap = O(n \log n)

• HeapSort:
  
n calls to swap and adjust = O(n \log n)

• Total:
  
O(n \log n)
Your Turn

• Worksheet 34 – BuildHeap and Heapsort