Nov. 6 – Heap Operations
Wade Fagen-Ulmschneider
template <class T>
void Heap<T>::_insert(const T & key) {
  // Check to ensure there's space to insert an element
  // ...if not, grow the array
  if ( size_ == capacity_ ) { _growArray(); } 

  // Insert the new element at the end of the array
  item_[++size] = key;

  // Restore the heap property
  _heapifyUp(size);
}
Exam Updates

Exam 9 (theory) is live!

Exam 10 is a programming exam:
• MPs: mp5
• Labs: lab_btree, lab_hash
• Lecture: Hashing Implementation (eg: Double Hashing)
  Heap Implementation
ICPC Regionals

UIUC’s ICPC Team at Regionals:
• We took 5 teams: our top three places 1st, 3rd, and 6th
• It’s not too late to join IPL
  • Mondays, 7pm – 9pm
removeMin
removeMin

template <class T>
void Heap<T>::_removeMin() {
    // Swap with the last value
    T minValue = item_[1];
    item_[1] = item_[size_];
    size--;
    // Restore the heap property
    heapifyDown();
    // Return the minimum value
    return minValue;
}
template <class T>
void Heap<T>::_removeMin() {
    // Swap with the last value
    T minValue = item_[1];
    item_[1] = item_[size_-];
    size--;

    // Restore the heap property
    _heapifyDown();

    // Return the minimum value
    return minValue;
}

template <class T>
void Heap<T>::_heapifyDown(int index) {
    if ( !_isLeaf(index) ) {
        T minChildIndex = _minChild(index);
        if ( item_[index] ___ item_[minChildIndex] ) {
            std::swap( item_[index], item_[minChildIndex] );
            _heapifyDown( ________________ );
        }
    }
}

Array Abstractions
buildHeap
1. Sort the array – it’s a heap!

2. template <class T>
   void Heap<T>::buildHeap() {
      for (unsigned i = 0; i <= size_; i++) {
         heapifyUp(i);
      }
   }

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

Strategy:
- We know that constant work is done based on the distance a node is away from the root (eg: it’s height).
- Therefore, the running time is proportional to the sum of the heights of the heights of all the nodes.
- We will work towards creating a proof around the sum of the heights of all the nodes.
Proving buildHeap Running Time

$S(h)$: Sum of the heights of all nodes in a complete tree of height $h$.

$S(0) =$

$S(1) =$

$S(h) =$
Proving buildHeap Running Time

Proof the recurrence:
  Base Case:

General Case:
Proving buildHeap Running Time

No one cares about things in terms of height:

\[ S(h): \]

Since \( h \leq \lg(n) \):

\[ \text{RunningTime}(n) \leq \]
Heap Sort

Running Time?

Why do we care about another sort?
A(nother) throwback to CS 173...

Let $R$ be an equivalence relation on $\text{us}$ where $(s, t) \in R$ if $s$ and $t$ have the same favorite among:

$$\{ __, __, __, __, __, __, \}$$
CS 225 – Things To Be Doing

Register for CS 225’s Final Exam!

Exam 9 (theory) is live!
More Info: https://courses.engr.illinois.edu/cs225/fa2017/exams/

MP5 is due tonight (grace period through tomorrow)
Due Monday, Nov. 6 at 11:59pm

New lab on Wednesday!
Due Sunday, Nov. 12 at 11:59pm

POTD
Every Monday-Friday – Worth +1 Extra Credit /problem (up to +40 total)