Recitation 8

Agenda
  - overhead
  - exercises
  - coding
  - (?) shell sort
  - psets

Overhead:
  - Psets:
    - look @ solns
    - stats
    - A mean: median:
    - B mean: median:
    - handout & end

Quiz:
  - 10/15 7:30-9:30 PM
  - Conflict 10/16 8-10 AM
    - email staff w/ reason
    - if previously emailed - confirm.

Feedback Form

Questions from lecture / anything you want reviewed.

= Pset
  - nlogn bound on \( T(n) = T(n^{1/2}) + n^{1/2} + c \) is not tight.
    - yes you can prove \( O(n \log n) \)
    - missing proof of \( \Omega(n \log n) \) \( \Rightarrow \) impossible.
      - real runtime \( \Theta(n) \)
      - why? b/c \( n^{1/2} \) does not add \( n \) per step!

  - OOPS - my apologies.
    \[ \log(f(n)) = \Theta(\log(g(n))) \Rightarrow f(n) = \Theta(g(n)) \] is FALSE.

    \[ \log(2^n) \text{ vs. } \log(3^n) \]
    \[ n \log(2) \text{ vs. } n \log(3) \]
    yes \( \Theta \) ... BUT you need to convert back.
    you can't cancel const values b/c
    they are actually \( e^c \).
Shell sort

idea: some sorts are very fast for nearly sorted data. We can nearly sort data really fast then pass off the data

sort every nth element, n decreases each time.

- lets elements make bigger "jumps" toward correct position

visually: use columns

then sort the columns.

gap sequence matters!

1 4 10 23 57 132 301 ...

algorithm on wikipedia

Worst case \( \Theta(n^2) \) with gap seq 1 2 4 8 ...

best \( \Theta(n \log^2 n) \)

Works well on real data.

Selection Sorts

idea: select an element, remove it, repeat.

if you select in a logical manner (max/min), you can sort!

Variables / options: storage data structure selection criteria.

<table>
<thead>
<tr>
<th>Struct</th>
<th>array</th>
<th>array</th>
<th>balanced BST</th>
<th>heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>min</td>
<td>max</td>
<td>min/max</td>
<td>min/max</td>
</tr>
<tr>
<td>time per select</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(lgn)</td>
<td>O(lgn)</td>
</tr>
<tr>
<td>total</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>O(nlgn)</td>
<td>O(nlgn)</td>
</tr>
</tbody>
</table>
EXERCISES

1. State a situation for which insertion sort would be a good choice of sort.

2. What is the worst possible sort you can think of? Each operation must be productive (i.e., no while (i < 3))

3. Analyze the runtimes of insertion + mergesort on lists that are:
   a. Sorted
   b. Reverse-sorted

4. How does insertion sort's runtime change if you use a linked list?

5. What is the runtime of the following code? \( \Theta(\quad) \)
   Then change the code to improve the runtime. You may assume all elements are \( \geq 0 \).

```python
def mergesort(A):
    if len(A) == 1:
        return A
    m = len(A) / 2
    B = len(A) * [-1]
    C = len(A) * [-1]
    B[:mid] = mergesort(A[:mid])
    C[:len(A)-mid] = mergesort(A[mid+1:])
    return merge(A, B, C)

def merge(A, B, C):
    a = 0
    b = 0
    c = 0
    while a < len(B) and b < len(A) and c < len(C):
        if B[b] <= C[c]:
            A[a] = B[b]
            b += 1
        else:
            A[a] = C[c]
            c += 1
        a += 1
```
6. convert the heap into its array representation

```
20
15 18
7 9 12 4
3 6 17 10
```

7. 

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>17</td>
</tr>
</tbody>
</table>
```

parent of x:
left (x):
right (x):
depth of heap:

parent (y):
left (y):
right (y):

8. restore the max-heap property

```
20
2
7 9
3 6 17 10
```

9. get code heaps.py from website (or take a handout)
a. implement left(i), right(i) and parent(i)
b. complete the missing code in heapify
   optional: what code would go here for a min-heap?
c. modify heapify to take a length argument.
d. implement delete-max
e. implement heap sort.
f. challenge: implement build which builds a heap in place from an arbitrary array.

if you change the method signatures - make sure to alter the tests accordingly!