Part 1: Priority Queue

Project 5

Priority Queue

- Implementing a priority queue through a binary heap.
- `PQ *createQueue(int (*compare)());`
  - return a pointer to a new priority queue using compare as its comparison function
- `void destroyQueue(PQ *pq);`
  - deallocate memory associated with the priority queue pointed to by pq
- `int numEntries(PQ *pq);`
  - return the number of entries in the priority queue pointed to by pq
- `void addEntry(PQ *pq, void *entry);`
  - add entry to the priority queue pointed to by pq
- `void *removeEntry(PQ *pq);`
  - remove and return the smallest entry from the priority queue pointed to by pq

Structure

```
struct pqueue {
        int count;              /* number of entries in array*/
        int length;             /* length of allocated array  */
        void **data;            /* allocated array of entries */
        int (*compare)();       /* comparison function        */
};
```

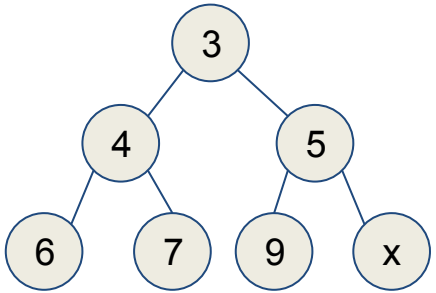
Dynamically allocated min heap

- Set initial length to 10, dynamically grow array in addEntry when needed
  - `realloc(target_pointer, new_size)`
- When adding, assume element starts at the end and reheap up
- When removing, save the root and replace it with the element at the end, then reheap down

addEntry and removeEntry

- `Parent = (((x) - 1) / 2)`
- `Left_child = ((x) * 2 + 1)`
- `Right_child = ((x) * 2 + 2)`
- Tip: When re-heaping, do not need to move the new entry until you find out where to put it

Heap Up Example



Inserting 2

Parent

Current (2 smaller than parent?)

3	4	5	6	7	9	x

Parent

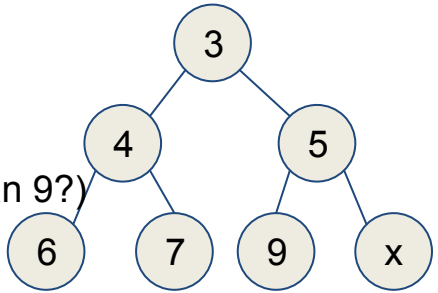
Current (2 smaller than parent?)

3	4	5	6	7	9	5

current

2	4	3	6	7	9	5

Heap Down Example



Removing Min

Current

Smallest Child (smaller than 9?)

3	4	5	6	7	9	x

Current

Smallest Child (smaller than 9?)

4	4	5	6	7	9	x

Current

Considered empty (larger than count)

4	6	5	9	7	9	x

Testing

- Generic data-type: use the comparison function given by the driver program
- Test with sort.c
  - Run with "`./sort`" and input more than 10 random numbers.
  - Works exactly like radix.c from the previous lab.
- Next week's lab (huffman tree) will use the priority queue from this week.