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 addElement when needed
- 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 reheaping, do not move the new entry until after you find out where to put it

Testing

- Generic data-type -> use the passed in comparison function

- Test with sort.c
  - Compile with “make sort”
  - Run with “./sort”
  - Works exactly like radix.c from the previous lab

- 2nd week’s lab will use 1st week’s lab!