

15-122: Principles of Imperative Computation, Spring 2020

Written Homework 10

Due on Gradescope: Thursday 9th April, 2020 by 9pm

Name: _____

Andrew ID: _____

Section: _____

This written homework covers priority queues and their implementation as heaps.

Preparing your Submission You can prepare your submission with any PDF editor that you like. Here are a few that prior-semester students recommended:

- *PDFescape* or *DocHub*, two web-based PDF editors that work from anywhere.
- *Preview*, the Mac's PDF viewer.
- *Acrobat Pro*, installed on all non-CS cluster machines, works on many platforms.
- *iAnnotate* works on any iOS and Android mobile device.

There are many more — use whatever works best for you. If you'd rather not edit a PDF, you can always print this homework, write your answers *neatly* by hand, and scan it into a PDF file — *we don't recommend this option, though*.

Submitting your Work Once you are done, submit this assignment on Gradescope. *Always check it was correctly uploaded.* You have unlimited submissions.

Question:	1	2	3	Total
Points:	5	3.5	3.5	12
Score:				

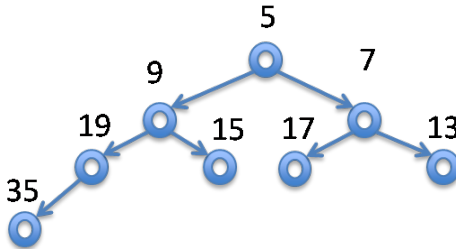
1. Heaps

As discussed in class, a *min-heap* is a hierarchical data structure that satisfies two invariants:

Order: Every child has value greater than or equal to its parent.

Shape: Each level of the min-heap is completely full except possibly the last level, which has all of its elements stored as far left as possible. (Also known as a *complete* binary tree).

Smaller integer values represent higher priorities. Consider:



0.5pts

- 1.1 Draw a picture of the final state of this min-heap after an element with value 11 is inserted. Satisfy the shape invariant first, then restore the order invariant while maintaining the shape invariant. Draw all branches in your tree *clearly* so we can distinguish left branches from right branches.

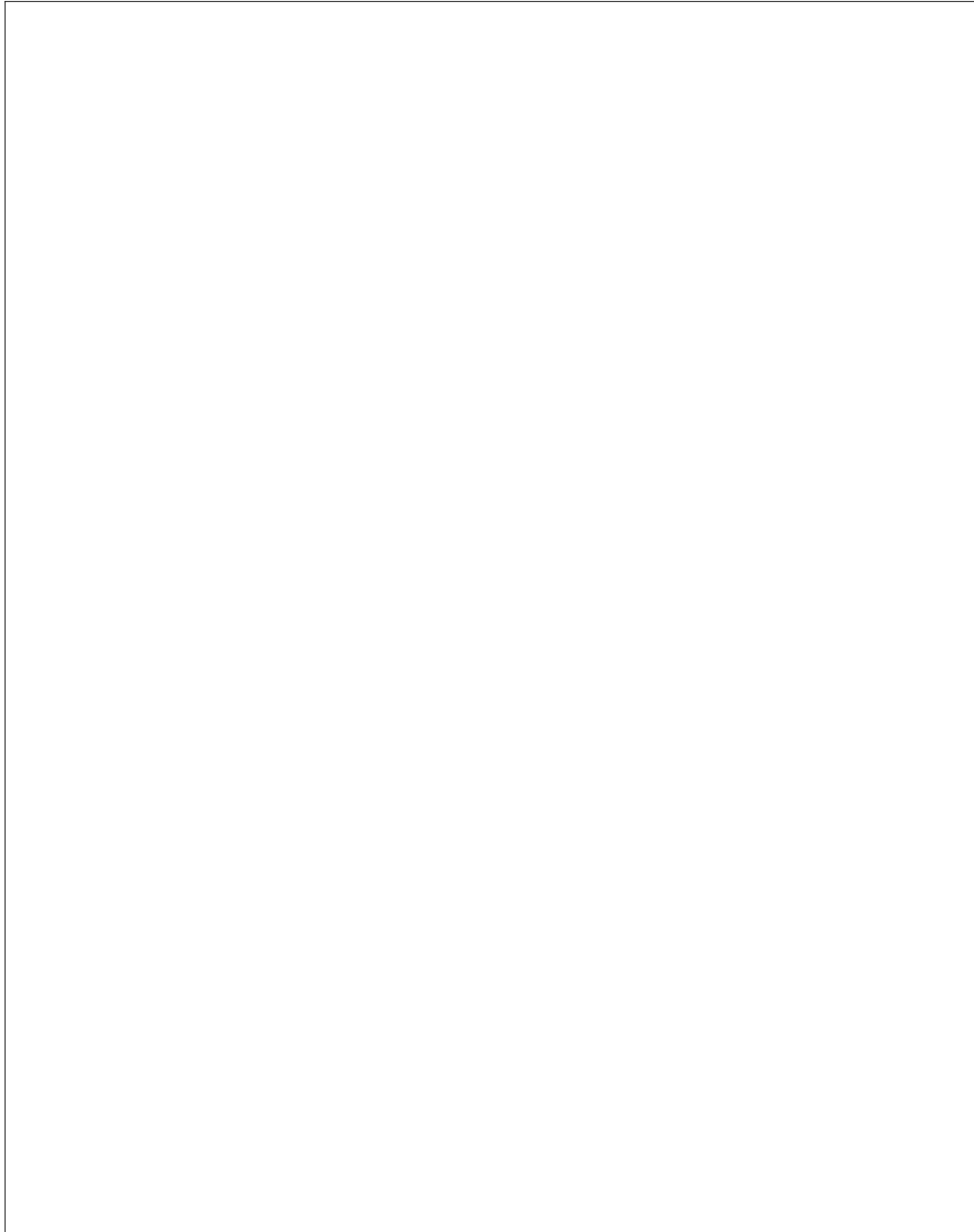
0.5pts

- 1.2 Starting from the *original* min-heap above, draw a picture of the final state of the min-heap after the element with the minimum value (i.e., with highest priority) is deleted. Satisfy the shape invariant first, then restore the order invariant while maintaining the shape invariant.

2.5pts

- 1.3 Insert the following values into an *initially empty* min-heap one at a time in the order shown. Draw the final state of the min-heap after each insert is completed and the min-heap is restored back to its proper invariants. Your answer should show 8 clearly drawn heaps.

27, 23, 40, 25, 7, 26, 44, 22



0.5pts

1.4 We are given an array A of n integers. Consider the following sorting algorithm:

- Insert every integer from A into a min-heap.
- Repeatedly delete the minimum from the heap, storing the deleted values back into A from left to right.

What is the worst-case runtime complexity of this sorting algorithm, using big- O notation? Briefly explain your answer.

O (_____)

Because:

0.5pts

1.5 You are given a non-empty min-heap. In one sentence, describe precisely where the maximum value must be located. Do not assume the heap is implemented as an array — your vocabulary should pertain only to the tree definition of a heap.

0.5pts

1.6 What is the worst-case runtime complexity of finding the maximum in a min-heap if the min-heap has n elements and is implemented as an array? Why?

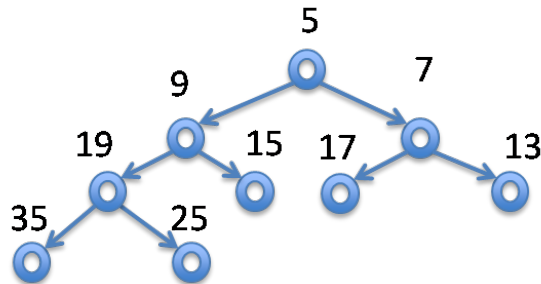
O (_____)

Because the number of values that need to be examined is _____.

2. Implementing Priority Queues as Arrays

0.5pts

2.1 Assume a priority queue is stored in an array as discussed in class. Using the min-heap pictured below, show where each element is stored in the array. You may not need to use all of the array positions shown below.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1pt

2.2 Consider a non-empty priority queue of n elements, each with a different priority. This priority queue is represented using the array implementation of min-heaps. Give the exact range (inclusive), in terms of n , of array indexes where any element of lowest priority might occur. You may use mathematical notation or C0 notation.

1pt

2.3 Complete the function `heap_lowest`, which returns (but does not remove) the element with the lowest priority from a min-heap stored as an array. The function template below is missing some loop invariants that would be needed to ensure safety; you don't have to add these.

You can use helper functions from the published `heap.c1` or functions in the client interface. *You shall examine only those elements that might contain the lowest priority element.*

```
elem heap_lowest(heap* H)
//@requires is_heap(H) && !pq_empty(H);
//@ensures is_heap(H);
{
    int index = _____;

    for (int i = _____; i < _____; i++) {

        if ( _____ ) {
            index = i;
        }
    }
    return _____;
}
```

1pt

2.4 Fill in the `@assert` directive as to support the safety of the implementation of the function `pq_rem`. You can find relevant code in file `heap.c1` published along the lecture notes. It removes and returns the element with the highest priority from a min-heap stored as an array.

```
elem pq_rem(heap H)
//@requires is_heap(H) && !pq_empty(H);
//@ensures is_heap(H);
{
    elem min = H->data[1];
    (H->next)--;

    if (H->next > 1) {
        H->data[1] = H->data[H->next];

        //@assert _____;
        sift_down(H);
    }
    return min;
}
```

3. Using Priority Queues

You are working an exciting desk job as a stock market analyst. You want to be able to determine the total price increase of the stocks that have seen the highest price increases over the last day (of course, on a bad day, these might simply be the least negative price changes). However, since the year is 1983, your Commodore 64 can only offer up about 30 KB of memory.

Stock reports are delivered to you via a `stream_t` data type with the following interface:

```
// typedef _____ stream_t;
typedef struct stock_report report;
struct stock_report {
    string company;
    int current_pps;           // stock price in cents
    int old_pps;              // previous day's price per share in cents
};

// Returns true if the data stream is empty
bool stream_empty(stream_t S);
// Retrieve the next stock report from the data stream
report* get_report(stream_t S) /*@requires !stream_empty(S); @*/ ;
```

A stream of stock reports could be very, very large. Storing all of the reports in an array won't cut it — you don't have enough memory (30 KB isn't even enough to store 2000 reports). You'll need a more clever solution.

Luckily, your cubicle mate Grace just finished a stellar priority queue implementation with the interface below. You think you should be able to use Grace's priority queue to keep track of only the stock reports on the stocks that have increased the most, discarding the others as necessary.

```
// Client Interface
// f(x,y) returns true if x is STRICTLY higher priority than y
typedef bool higher_priority_fn(void* x, void* y);

// Library Interface
// typedef _____* pq_t;
pq_t pq_new(int capacity, higher_priority_fn* priority)
    /*@requires capacity > 0 && priority != NULL; @*/
    /*@ensures \result != NULL; @*/ ;
bool pq_full(pq_t Q)           /*@requires Q != NULL; @*/ ;
bool pq_empty(pq_t Q)         /*@requires Q != NULL; @*/ ;
void pq_add(pq_t Q, void* x) /*@requires Q != NULL && !pq_full(Q); @*/
    /*@requires x != NULL; @*/ ;
void* pq_rem(pq_t Q)          /*@requires Q != NULL && !pq_empty(Q); @*/ ;
void* pq_peek(pq_t Q)         /*@requires Q != NULL && !pq_empty(Q); @*/ ;
```

2.5pts

- 3.1 Complete the functions `client_priority` and `total_increase` below. The function `total_increase` returns the sum of the pps increases of the `n` stocks with the highest pps increases from the data stream `S`.

```

#use <util>

bool client_priority(void* x, void* y)
//@requires x != NULL && \hastag(report*, x);
//@requires y != NULL && \hastag(report*, y);
{
    return _____;
}

int total_increase(stream_t S, int n)
//@requires 0 < n && n < int_max();
{
    pq_t Q = pq_new(_____);

    while (!stream_empty(S)) {
        // Put the next stock report into the priority queue
        _____;
        // If the priority queue is at capacity, delete the
        // report with the smallest pps increase

        if ( _____ )
            _____;
    }

    // Add up the pps increases of everything in the
    // priority queue
    int total = 0;

    while ( _____ ) {
        report* r = _____;
        total += _____;
    }

    return total;
}

```


1pt

3.2 Assuming that `get_report` is a constant-time function and that Grace's priority queues are based on the heap data structure, what is the running time of `total_increase(S, n)` if the stream `S` ultimately contains m elements? (Give an answer in big-O notation.)

$O(\rule{15em}{0.4pt})$
