CMSC 420:Fall 2019 Dave Mount

Practice Problems for the Final Exam

Final Exam will be in class on **Wed, Dec 11, 8:00-10:00am**. The exam will be closed-book, closed-notes, but you will be allowed *3 sheets of notes*, front and back (handwritten or typeset, your choice). Please plan to bring your university ID with you during the exam.

Disclaimer: This just reflects the material since the second midterm. These practice problems have been extracted from old homework assignments and exams. Material changes from semester to semester. These do **not** necessarily reflect the actual coverage, difficulty, or length of the midterm exam.

Problem 1. Since the exam is comprehensive, please look over all the old homework assignments, exams, and practice problems.

Problem 2. Recall the buddy system of allocating blocks of memory (see Fig. 1). Throughout this problem you may use the following standard bit-wise operators:

`&`	bit-wise "and"	`|`	bit-wise "or"
`^`	bit-wise "exclusive-or"	`~`	bit-wise "complement"
`<<`	left shift (filling with zeros)	`>>`	right shift (filling with zeros)

You may also assume that you have access to a function `bitMask(k)`, which returns a binary number whose k lowest-order bits are all 1's. For example `bitMask(3)` $= 111_2 = 7$.

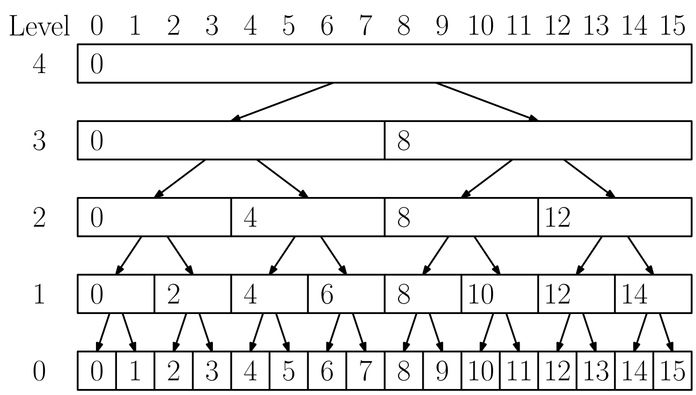

Figure 1: Buddy relatives.

Present a short (one-line) expression for each of the following functions in terms of the above bit-wise functions:

(i) `boolean isValid(int k, int x)`: True if and only if $x \geq 0$ a valid starting address for a buddy block at level $k \geq 0$.

(ii) `int sibling(int k, int x)`: Given a valid buddy block of level $k \geq 0$ starting at address x, returns the starting address of its *sibling*.

(iii) `int parent(int k, int x)`: Given a valid buddy block of level $k \geq 0$ starting at address x, returns the starting address of its *parent* at level $k+1$.

(iv) `int left(int k, int x)`: Given a valid buddy block of level $k \geq 1$ starting at address x, returns the starting address of its *left child* at level $k-1$.

(v) `int right(int k, int x)`: Given a valid buddy block of level $k \geq 1$ starting at address x, returns the starting address of its *right child* at level $k-1$.

Problem 3. Suppose you have a large span of memory, which starts at some address `start` and ends at address `end-1` (see Fig. 2). (The variables `start` and `end` are generic pointers of type `void*`.) As the dynamic memory allocation method of Lecture 15, this span is subdivided into blocks. The block starting at address `p` is associated with the following information:

- `p.inUse` is 1 if this block is in-use (allocated) and 0 otherwise (available)
- `p.prevInUse` is 1 if the block immediately preceeding this block in memory is in-use. (It should be 1 for the first block.)
- `p.size` is the number of words in this block (including all header fields)
- `p.size2` each available block has a copy of the size stored in its last word, which is located at address `p + p.size - 1`.

(For this problem, we will ignore the available-list pointers `p.prev` and `p.next`.)

In class, we said that in real memory-allocation systems, blocks cannot be moved, because they may contain pointers. Suppose, however, that the blocks are movable. Present pseudo-code for a function that compacts memory by copying all the allocated blocks to a single contiguous span of blocks at the start of the memory span (see Fig. 2). Your function `compress(void* start, void* end)` should return a pointer to the head of the available block at the end. Following these blocks is a single available block that covers the rest of the memory's span.

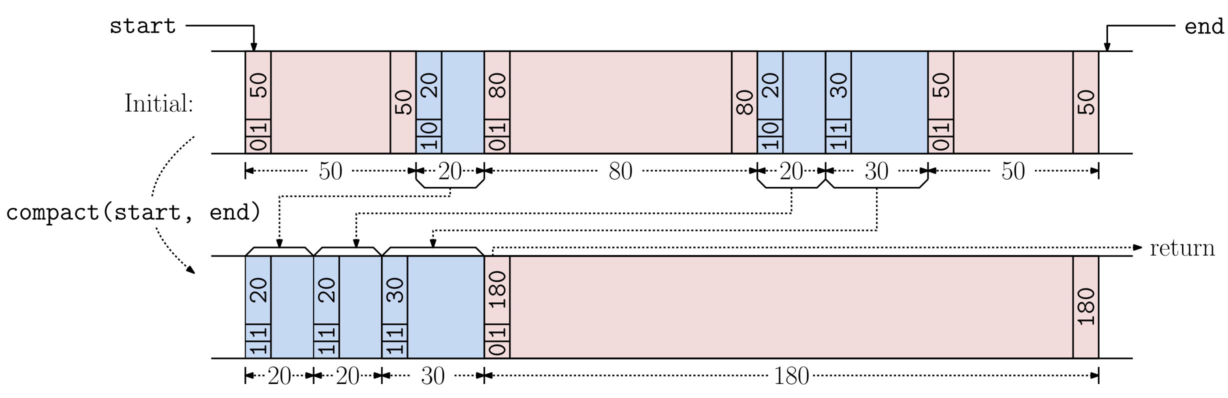

Figure 2: Memory compactor.

To help copy blocks of memory around, you may assume that you have access to a function `void* memcpy(void* dest, void* source, int num)`, which copies `num` words of memory from the address `source` to the address `dest`.