CSE 220: Systems Fundamentals I

Stony Brook University

Programming Project #4

Spring 2020

Assignment Due: Sunday, April 26, 2020 by 11:59 pm

Updates to the Document:

- 4/14/2020: updated Part 2 to clarify that no changes must be written to memory if `max_size` is invalid.
- 4/14/2020: a correction to Part 5, Example 3 was made.
- 4/20/2020: clarified pseudocode in Part 8 for a special case.

Learning Outcomes

After completion of this programming project you should be able to:

- Implement non-trivial algorithms that require conditional execution and iteration.
- Design and code functions that implement the MIPS assembly register conventions.
- Implement algorithms that process arrays of structs.

Getting Started

Visit Piazza and download the file `proj4.zip`. Decompress the file and then open `proj4.zip`. Fill in the following information at the top of `proj4.asm`:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)

Having this information at the top of the file helps us locate your work. If you forget to include this information but don't remember until after the deadline has passed, don't worry about it – we will track down your submission.

Inside `proj4.asm` you will find several function stubs that consist simply of `jr $ra` instructions. Your job in this assignment is implement all the functions as specified below. Do not change the function names, as the grading scripts will be looking for functions of the given names. However, you may implement additional helper functions of your own, but they must be saved in `proj4.asm`. Helper functions will not be graded.

If you are having difficulty implementing these functions, write out pseudocode or implement the functions in a higher-level language first. Once you understand the algorithm and what steps to perform, then translate the logic into MIPS assembly code.

Be sure to initialize all of your values (e.g., registers) within your functions. Never assume registers or memory will hold any particular values (e.g., zero). MARS initializes all of the registers and bytes of main memory to zeroes. The grading scripts will fill the registers and/or main memory with random values before calling your functions.

Finally, do not define a `.data` section in your `proj4.asm` file. A submission that contains a `.data` section will probably receive a score of zero.

Important Information about CSE 220 Homework Assignments

- Read the entire homework documents twice before starting. Questions posted on Piazza whose answers are clearly stated in the documents will be given lowest priority by the course staff.
- **You must use the Stony Brook version of MARS posted on Blackboard.** Do not use the version of MARS posted on the official MARS website. The Stony Brook version has a reduced instruction set, added tools, and additional system calls you might need to complete the homework assignments.
- When writing assembly code, try to stay consistent with your formatting and to comment as much as possible. It is much easier for your TAs and the professor to help you if we can quickly figure out what your code does.
- You personally must implement the programming projects in MIPS Assembly language by yourself. You may not write or use a code generator or other tools that write any MIPS code for you. You must manually write all MIPS assembly code you submit as part of the assignments.
- Do not copy or share code. Your submissions will be checked against other submissions from this semester and from previous semesters.
- Do not submit a file with the function/label `main` defined. You are also not permitted to start your label names with two underscores (__). You will obtain a zero for an assignment if you do this.
- Submit your final `.asm` file to Blackboard by the due date and time. Late work will not be accepted or graded. Code that crashes and cannot be graded will earn no credit. No changes to your submission will be permitted once the deadline has passed.

How Your CSE 220 Assignments Will Be Graded

With minor exceptions, all aspects of your homework submissions will be graded entirely through automated means. Grading scripts will execute your code with input values (e.g., command-line arguments, function arguments) and will check for expected results (e.g., print-outs, return values, etc.) For this homework assignment you will be writing *functions* in assembly language. The functions will be tested independently of each other. This is very important to note, as you must take care that no function you write ever has side-effects or requires that other functions be called before the function in question is called. Both of these are generally considered bad practice in programming.

Some other items you should be aware of:

- Each test case must execute in 500,000 instructions or fewer. Efficiency is an important aspect of programming. This maximum instruction count will be increased in cases where a complicated algorithm might be necessary, or a large data structure must be traversed. To find the instruction count of your code in MARS, go to the **Tools** menu and select **Instruction Statistics**. Press the button marked **Connect to MIPS**. Then assemble and run your code as normal.

- Any excess output from your program (debugging notes, etc.) might impact grading. Do not leave erroneous print-outs in your code.
- We will provide you with a small set of test cases for each assignment to give you a sense of how your work will be graded. It is your responsibility to test your code thoroughly by creating your own test cases.
- The testing framework we use for grading your work will not be released, but the test cases and expected results used for testing will be released.

Register Conventions

You must follow the register conventions taught in lecture and reviewed in recitation. Failure to follow them will result in loss of credit when we grade your work. Here is a brief summary of the register conventions and how your use of them will impact grading:

- It is the callee's responsibility to save any `$s` registers it overwrites by saving copies of those registers on the stack and restoring them before returning.
- If a function calls a secondary function, the caller must save `$ra` before calling the callee. In addition, if the caller wants a particular `$a`, `$t` or `$v` register's value to be preserved across the secondary function call, the best practice would be to place a copy of that register in an `$s` register before making the function call.
- A function which allocates stack space by adjusting `$sp` must restore `$sp` to its original value before returning.
- Registers `$fp` and `$gp` are treated as preserved registers for the purposes of this course. If a function modifies one or both, the function must restore them before returning to the caller. There really is no reason for your code to touch the `$gp` register, so leave it alone.

The following practices will result in loss of credit:

- "Brute-force" saving of all `$s` registers in a function or otherwise saving `$s` registers that are not overwritten by a function.
- Callee-saving of `$a`, `$t` or `$v` registers as a means of "helping" the caller.
- "Hiding" values in the `$k`, `$f` and `$at` registers or storing values in main memory by way of offsets to `$gp`. This is basically cheating or at best, a form of laziness, so don't do it. We will comment out any such code we find.

How to Test Your Functions

To test your implemented functions, open the provided `main` files in MARS. Next, assemble the `main` file and run it. MARS will include the contents of any `.asm` files referenced with the `.include` directive(s) at the end of the file and then enqueue the contents of your `proj4.asm` file before assembling the program.

Each main file calls a single function with one of the sample test cases and prints any return value(s). You will need to change the arguments passed to the functions to test your functions with the other cases. To test each of your functions thoroughly, create your own test cases in those main files. Your submission will not be graded using the examples provided in this document or using the provided main file(s). Do not submit your main files to Blackboard – we will delete them. Also please note that these testing main files will not be used in grading.

Again, any modifications to the `main` files will not be graded. You will submit only your `proj4.asm` for grading. Make sure that all code required for implementing your functions is included in the `proj4.asm` file. To make sure that your code is self-contained, try assembling your `proj4.asm` file by itself in MARS. If you get any errors (such as a missing label), this means that you need to refactor (reorganize) your code, possibly by moving labels you inadvertently defined in a `main` file (e.g., a helper function) to `proj4.asm`.

A Reminder on How Your Work Will be Graded

It is **imperative** (crucial, essential, necessary, critically important) that you implement the functions below exactly as specified. Do not deviate from the specifications, even if you think you are implementing the program in a better way. Modify the contents of memory only as described in the function specifications!

Priority Queues

For this assignment you will be implementing a priority queue using a ternary max-heap, as opposed to a binary max-heap, as you might have learned in your data structures course. In a ternary max-heap, values are arranged in a ternary (3-way) tree, in which each node is greater than or equal to all of its children.

In this assignment, you will implement operations to support a max-heap of customers waiting to get into a restaurant. A customer is implemented as a *struct*, which is a means in the C language for creating data structures. A priority queue will also be implemented as a struct.

```
struct Customer {
    int id;           # 32-bit unsigned word. Bytes 0-3 of the struct.
    short fame;       # 16-bit unsigned half-word. Bytes 4-5 of the struct.
    short wait_time;  # 16-bit unsigned half-word. Bytes 6-7 of the struct.
}

struct CustomerQueue {
    short size;           # 16-bit unsigned half-word. Bytes 0-1 of the struct.
    short max_size;       # 16-bit unsigned half-word. Bytes 2-3 of the struct.
    Customer[] customers; # ternary max-heap of "size" Customers.
}
```

We will always assume that `Customer` structs and `CustomerQueue` structs are word-aligned. When you want to read/write the 16-bit values in each struct, make things easy for yourself and use the `lhu` and `sh` instructions.

A `Customer` struct's `fame` measures a person's relative fame; higher values correspond with more famous people. The `wait_time` is in minutes.

The `customers` array inside a `CustomerQueue` struct is **not** an array of pointers to `Customer` structs, but actual `Customer` structs. Thus, each element in the `customers` array consumes 8 bytes of memory. The `size` field records how many `Customer` structs are stored in the heap. The `max_size` field indicates the maximum size of the heap. For example if `size = 4` and `max_size = 6`, this means that indices 0 through 3 of `customers` contain valid `Customer` structs and are arranged in a ternary max-heap. The elements at indices 4 and 5 should be treated as garbage values.

In a ternary tree implemented as an array (e.g., `customers`), any children of the node at index i are available at indices $3i+1$, $3i+2$ and $3i+3$. Since `customers` implements a max-heap, these children are guaranteed to be less than or equal to their parent. Note that for a node at index i in the array, its parent is located at index $(i-1)/3$, assuming integer division.

For our application, priority is a function of a customer's fame and waiting time:

$$priority = fame + 10 \times wait\_time$$

Ideally, we want famous people eating at our restaurant, but at the same time we don't want to tick off less-famous people who have been waiting for a long time.

Below is an example of a priority queue, where the indentation level helps to show the overall structure of the ternary max-heap, level-by-level. The three-digit numbers on the left provide the index of each customer in the `customers` array:

```
Size: 15
Max size: 18
000: id: 365, fame: 889, wait_time: 11 (priority: 999)
    001: id: 519, fame: 346, wait_time: 21 (priority: 556)
    002: id: 815, fame: 730, wait_time: 24 (priority: 970)
    003: id: 281, fame: 749, wait_time: 15 (priority: 899)
        004: id: 317, fame: 258, wait_time: 13 (priority: 388)
        005: id: 994, fame: 104, wait_time: 26 (priority: 364)
        006: id: 667, fame: 330, wait_time: 8 (priority: 410)
        007: id: 318, fame: 565, wait_time: 4 (priority: 605)
        008: id: 38, fame: 431, wait_time: 23 (priority: 661)
        009: id: 179, fame: 346, wait_time: 21 (priority: 556)
        010: id: 135, fame: 81, wait_time: 27 (priority: 351)
        011: id: 400, fame: 327, wait_time: 25 (priority: 577)
        012: id: 887, fame: 182, wait_time: 25 (priority: 432)
            013: id: 5, fame: 181, wait_time: 2 (priority: 201)
            014: id: 103, fame: 1, wait_time: 27 (priority: 271)
            015: garbage
            016: garbage
            017: garbage
```

Part 1: Compare Two Customer Structs

```
int compare_to(Customer* c1, Customer* c2)
```

This function takes two pointers to `Customer` structs and returns `-1` if `c1 < c2`, `0` if `c1 == c2` or `1` if `c1 > c2`. The "less than" operator's meaning is defined below in pseudocode.

The function takes the following arguments, in this order:

- `c1`: a pointer to a `Customer` struct
- `c2`: a pointer to a `Customer` struct

The algorithm to implement is as follows:

```
if c1.fame + 10 * c1.wait_time < c2.fame + 10 * c2.wait_time then
    return -1
else if c1.fame + 10 * c1.wait_time > c2.fame + 10 * c2.wait_time then
    return 1
```

```
else
    if c1.fame < c2.fame then
        return -1
    else if c1.fame > c2.fame then
        return 1
    else
        return 0
```

So, please note, that we are not implementing "priority" in a typical way here. Although we compute a priority value for each customer, in the case of a tie in priority, a more famous person is considered higher importance/priority, ultimately, for the purpose of inserting items into the max-heap.

For any function in this assigment where you need to compare two customers' priorities, always use `compare_to`. *Never* directly compare the priority metrics of the two customers.

Returns in `$v0`:

- `-1`, `0` or `1`, as described above.

Additional requirements:

- The function must not write any changes to main memory.

See the file `compare_to_args.asm` *for the* `.data` *sections for the examples below. Such a file is provided for most (not all) parts of the assignment.*

Example #1:

```
c1 = id: 1, fame: 100, wait_time: 10 (priority: 200)
c2 = id: 2, fame: 50, wait_time: 30 (priority: 350)
```

Return value: `-1`

Example #2:

```
c1 = id: 1, fame: 200, wait_time: 10 (priority: 300)
c2 = id: 2, fame: 50, wait_time: 5 (priority: 100)
```

Return value: `1`

Example #3:

```
c1 = id: 1, fame: 5, wait_time: 10 (priority: 105)
c2 = id: 2, fame: 105, wait_time: 0 (priority: 105)
```

Return value: `-1`

Example #3 is a case where, although both individuals have the same computed priority value, `c2` *would be closer to the front of the queue due to the person's higher fame.*

Example #4:

```
c1 = id: 1, fame: 5, wait_time: 10 (priority: 105)
```

```
c2 = id: 2, fame: 5, wait_time: 10 (priority: 105)
```

Return value: `0`

Part 2: Initialize an Empty Priority Queue

```
int init_queue(CustomerQueue* queue, int max_size)
```

This function initializes a new `CustomerQueue` struct located at the memory address referenced by `queue`. The function sets the `size` field of the struct to 0, the `max_size` field to the value of the argument `max_size`, and zeroes-out all bytes of the `queue.customers` array. Assume that enough memory has been allocated to store the queue.

The function takes the following arguments, in this order:

- `queue`: a pointer to an uninitialized `CustomerQueue`. Assume the allocated memory is full of garbage.
- `max_size`: the number of 8-byte elements of the queue's `customers` queue to zero-out

Returns in `$v0`:

- `max_size` if `max_size` $>$ 0; or
- `-1` if `max_size` $\leq$ 0

Additional requirements:

- The function must not write any changes to main memory except as necessary.
- If `max_size` is invalid, no changes at all may be written to memory.

Part 3: Copy a Memory Buffer

```
int memcpy(byte* src, byte* dest, int n)
```

This function copies `n` bytes of data from address `src` to address `dest`. You may assume that the `dest` buffer is at least `n` bytes in size. This function will be used in later functions to copy `Customer` structs from one place in memory to another. However, you should code for generality and not be concerned about the significance of the bytes that the function copies.

The function takes the following arguments, in this order:

- `src`: the address to copy bytes from
- `dest`: the address to copy bytes to
- `n`: the number of bytes to copy (must be greater than 0)

Note that `src` and `dest` are not necessarily word-aligned.

Returns in `$v0`:

- `n` if `n` $>$ 0; or
- `-1` if `n` $\leq$ 0

Additional requirements:

- The function must not write any changes to main memory except as necessary.
- Only the first `n` bytes memory starting at `dest` may be changed. All other bytes in `dest` must be left unchanged.

Examples:

Arguments	**Return Value**	**Updated** `dest`
`"ABCDEFGHIJ", "XXXXXXX", 3`	`3`	`"ABCXXXX"`
`"ABCDEFGHIJ", "XXXXXXX", 7`	`7`	`"ABCDEFG"`
`"ABCDEFGHIJ", "XXXXXXX", -3`	`-1`	`"XXXXXXX"`
`"ABCDEFGHIJ", "XXXXXXX", 0`	`-1`	`"XXXXXXX"`

Part 4: Test if a Customer is Present in a Queue

`int contains(CustomerQueue* queue, int customer_id)`

This function inspects a `CustomerQueue` struct and determines if a `Customer` with the given ID # is present in the queue. You may assume that every customer ID # in the queue is unique. If a customer with that ID # is present, the function returns the level of the ternary heap where the corresponding node is found. The root node's level is 1. If no customer with the given ID # is found, the function returns `-1`. *Recursion is not required to solve this problem.*

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. The queue might be empty.
- `customer_id`: the ID # of the `Customer` struct we are searching for

Returns in `$v0`:

- the level of `Customer` node with ID # `customer_id` in the ternary max-heap inside of `queue`, assuming that such a customer exists in the queue; or
- `-1`: if no such customer with ID # `customer_id` is found in the queue.

Additional requirements:

- The function must not write any changes to main memory.

Example #1:

```
queue =
    Size: 5
    Max size: 8
    000: id: 111, fame: 859, wait_time: 28 (priority: 1139)
        001: id: 550, fame: 576, wait_time: 10 (priority: 676)
        002: id: 788, fame: 418, wait_time: 8 (priority: 498)
        003: id: 896, fame: 40, wait_time: 12 (priority: 160)
            004: id: 54, fame: 275, wait_time: 2 (priority: 295)
customer_id = 788
```

Return value: `2`

Example #2:

```
queue =
    Size: 15
    Max size: 15
    000: id: 130, fame: 810, wait_time: 29 (priority: 1100)
        001: id: 349, fame: 873, wait_time: 1 (priority: 883)
        002: id: 105, fame: 928, wait_time: 1 (priority: 938)
        003: id: 90, fame: 900, wait_time: 19 (priority: 1090)
            004: id: 38, fame: 788, wait_time: 9 (priority: 878)
            005: id: 928, fame: 447, wait_time: 24 (priority: 687)
            006: id: 387, fame: 72, wait_time: 0 (priority: 72)
            007: id: 575, fame: 342, wait_time: 22 (priority: 562)
            008: id: 324, fame: 750, wait_time: 14 (priority: 890)
            009: id: 95, fame: 684, wait_time: 4 (priority: 724)
            010: id: 252, fame: 168, wait_time: 0 (priority: 168)
            011: id: 22, fame: 299, wait_time: 29 (priority: 589)
            012: id: 441, fame: 588, wait_time: 15 (priority: 738)
                013: id: 164, fame: 139, wait_time: 10 (priority: 239)
                014: id: 272, fame: 481, wait_time: 26 (priority: 741)
customer_id = 441
```

Return value: `3`

Example #3:

```
queue =
    Size: 28
    Max size: 35
    000: id: 579, fame: 829, wait_time: 30 (priority: 1129)
        001: id: 915, fame: 788, wait_time: 20 (priority: 988)
        002: id: 14, fame: 918, wait_time: 11 (priority: 1028)
        003: id: 624, fame: 655, wait_time: 16 (priority: 815)
            004: id: 744, fame: 840, wait_time: 2 (priority: 860)
            005: id: 346, fame: 568, wait_time: 12 (priority: 688)
            006: id: 852, fame: 644, wait_time: 11 (priority: 754)
            007: id: 514, fame: 742, wait_time: 26 (priority: 1002)
            008: id: 683, fame: 611, wait_time: 28 (priority: 891)
            009: id: 854, fame: 498, wait_time: 16 (priority: 658)
            010: id: 685, fame: 430, wait_time: 11 (priority: 540)
            011: id: 523, fame: 33, wait_time: 27 (priority: 303)
            012: id: 773, fame: 661, wait_time: 5 (priority: 711)
                013: id: 323, fame: 13, wait_time: 12 (priority: 133)
                014: id: 936, fame: 499, wait_time: 21 (priority: 709)
                015: id: 269, fame: 175, wait_time: 10 (priority: 275)
                016: id: 495, fame: 212, wait_time: 26 (priority: 472)
                017: id: 906, fame: 76, wait_time: 23 (priority: 306)
                018: id: 468, fame: 397, wait_time: 18 (priority: 577)
                019: id: 798, fame: 345, wait_time: 0 (priority: 345)
                020: id: 194, fame: 601, wait_time: 2 (priority: 621)
                021: id: 526, fame: 445, wait_time: 21 (priority: 655)
```

```
                 022: id: 352, fame: 535, wait_time: 25 (priority: 785)
                 023: id: 235, fame: 416, wait_time: 17 (priority: 586)
                 024: id: 552, fame: 654, wait_time: 25 (priority: 904)
                 025: id: 931, fame: 62, wait_time: 20 (priority: 262)
                 026: id: 295, fame: 507, wait_time: 30 (priority: 807)
                 027: id: 783, fame: 646, wait_time: 23 (priority: 876)
customer_id = 295
```

Return value: `4`

Example #4:

```
queue =
    Size: 5
    Max size: 8
    000: id: 239, fame: 558, wait_time: 15 (priority: 708)
        001: id: 138, fame: 468, wait_time: 6 (priority: 528)
        002: id: 456, fame: 388, wait_time: 24 (priority: 628)
        003: id: 473, fame: 46, wait_time: 0 (priority: 46)
            004: id: 182, fame: 158, wait_time: 17 (priority: 328)
customer_id = 22
```

Return value: `-1`

Part 5: Enqueue a Customer

`(int, int) enqueue(CustomerQueue* queue, Customer* customer)`

This function enqueues the given customer into the given priority queue, maintaining the max-heap organization of the `customers` array. You will need to adapt the insert (enqueue) operation for binary heaps depicted on the Wikipedia page for binary heaps. Briefly, call `memcpy` to copy `customer` to index `queue.size` of the `queue.customers` array. Then, iteratively swap the node with its parent while the node's priority (as implemented by `compare_to`) is strictly greater than its parent. The function adds 1 to the `size` field of the queue.

There could be two reasons why the enqueue operation cannot be performed:

- a customer in the queue with customer ID# `customer.id` already exists in the queue; or
- the queue is already full

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. It is possible that the queue is empty or full.
- `customer`: a pointer to a valid `Customer` struct

Returns in `$v0`:

- `1`: if the insertion was successful, or
- `-1`: if the insertion could not be completed for one of the reasons cited above

Returns in `$v1`:

- the size of the queue after the attempted insertion has completed. In error cases, the size must remain unchanged.

Additional requirements:

- The function must call `contains`, `compare_to`, and `memcpy` to copy the new customer into the queue and as part of the heapify-up process.
- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 8
    Max size: 12
    000: id: 111, fame: 859, wait_time: 28 (priority: 1139)
        001: id: 349, fame: 873, wait_time: 1 (priority: 883)
        002: id: 575, fame: 342, wait_time: 22 (priority: 562)
        003: id: 896, fame: 40, wait_time: 12 (priority: 160)
            004: id: 54, fame: 275, wait_time: 2 (priority: 295)
            005: id: 550, fame: 576, wait_time: 10 (priority: 676)
            006: id: 164, fame: 139, wait_time: 10 (priority: 239)
            007: id: 788, fame: 418, wait_time: 8 (priority: 498)
customer = id: 252, fame: 668, wait_time: 0 (priority: 668)
```

Updated queue:

```
    Size: 9
    Max size: 12
    000: id: 111, fame: 859, wait_time: 28 (priority: 1139)
        001: id: 349, fame: 873, wait_time: 1 (priority: 883)
        002: id: 252, fame: 668, wait_time: 0 (priority: 668)
        003: id: 896, fame: 40, wait_time: 12 (priority: 160)
            004: id: 54, fame: 275, wait_time: 2 (priority: 295)
            005: id: 550, fame: 576, wait_time: 10 (priority: 676)
            006: id: 164, fame: 139, wait_time: 10 (priority: 239)
            007: id: 788, fame: 418, wait_time: 8 (priority: 498)
            008: id: 575, fame: 342, wait_time: 22 (priority: 562)
```

Return values: `1, 9`

Example #2:

```
queue =
    Size: 25
    Max size: 30
    000: id: 579, fame: 829, wait_time: 30 (priority: 1129)
        001: id: 130, fame: 810, wait_time: 29 (priority: 1100)
        002: id: 744, fame: 840, wait_time: 2 (priority: 860)
        003: id: 324, fame: 750, wait_time: 14 (priority: 890)
            004: id: 90, fame: 900, wait_time: 19 (priority: 1090)
            005: id: 915, fame: 788, wait_time: 20 (priority: 988)
            006: id: 105, fame: 928, wait_time: 1 (priority: 938)
            007: id: 272, fame: 481, wait_time: 26 (priority: 741)
            008: id: 22, fame: 299, wait_time: 29 (priority: 589)
            009: id: 441, fame: 588, wait_time: 15 (priority: 738)
            010: id: 38, fame: 788, wait_time: 9 (priority: 878)
            011: id: 352, fame: 535, wait_time: 25 (priority: 785)
            012: id: 495, fame: 212, wait_time: 26 (priority: 472)
                013: id: 928, fame: 447, wait_time: 24 (priority: 687)
                014: id: 624, fame: 655, wait_time: 16 (priority: 815)
                015: id: 323, fame: 13, wait_time: 12 (priority: 133)
                016: id: 95, fame: 684, wait_time: 4 (priority: 724)
                017: id: 526, fame: 445, wait_time: 21 (priority: 655)
                018: id: 552, fame: 654, wait_time: 25 (priority: 904)
                019: id: 683, fame: 611, wait_time: 28 (priority: 891)
                020: id: 854, fame: 498, wait_time: 16 (priority: 658)
                021: id: 685, fame: 430, wait_time: 11 (priority: 540)
                022: id: 387, fame: 72, wait_time: 0 (priority: 72)
                023: id: 523, fame: 33, wait_time: 27 (priority: 303)
                024: id: 773, fame: 661, wait_time: 5 (priority: 711)
customer = id: 936, fame: 499, wait_time: 21 (priority: 709)
```

Updated queue:

```
    Size: 26
    Max size: 30
    000: id: 579, fame: 829, wait_time: 30 (priority: 1129)
        001: id: 130, fame: 810, wait_time: 29 (priority: 1100)
        002: id: 744, fame: 840, wait_time: 2 (priority: 860)
        003: id: 324, fame: 750, wait_time: 14 (priority: 890)
            004: id: 90, fame: 900, wait_time: 19 (priority: 1090)
            005: id: 915, fame: 788, wait_time: 20 (priority: 988)
            006: id: 105, fame: 928, wait_time: 1 (priority: 938)
            007: id: 272, fame: 481, wait_time: 26 (priority: 741)
            008: id: 936, fame: 499, wait_time: 21 (priority: 709)
            009: id: 441, fame: 588, wait_time: 15 (priority: 738)
            010: id: 38, fame: 788, wait_time: 9 (priority: 878)
            011: id: 352, fame: 535, wait_time: 25 (priority: 785)
            012: id: 495, fame: 212, wait_time: 26 (priority: 472)
                013: id: 928, fame: 447, wait_time: 24 (priority: 687)
                014: id: 624, fame: 655, wait_time: 16 (priority: 815)
                015: id: 323, fame: 13, wait_time: 12 (priority: 133)
                016: id: 95, fame: 684, wait_time: 4 (priority: 724)
                017: id: 526, fame: 445, wait_time: 21 (priority: 655)
                018: id: 552, fame: 654, wait_time: 25 (priority: 904)
                019: id: 683, fame: 611, wait_time: 28 (priority: 891)
                020: id: 854, fame: 498, wait_time: 16 (priority: 658)
                021: id: 685, fame: 430, wait_time: 11 (priority: 540)
                022: id: 387, fame: 72, wait_time: 0 (priority: 72)
                023: id: 523, fame: 33, wait_time: 27 (priority: 303)
                024: id: 773, fame: 661, wait_time: 5 (priority: 711)
                025: id: 22, fame: 299, wait_time: 29 (priority: 589)
```

Return values: `1, 26`

Example #3:

```
queue =
    Size: 5
    Max size: 6
    000: id: 346, fame: 568, wait_time: 12 (priority: 688)
        001: id: 798, fame: 345, wait_time: 0 (priority: 345)
        002: id: 906, fame: 76, wait_time: 23 (priority: 306)
        003: id: 468, fame: 397, wait_time: 18 (priority: 577)
            004: id: 269, fame: 175, wait_time: 10 (priority: 275)
customer = id: 906, fame: 76, wait_time: 23 (priority: 306)
```

Updated queue:

```
    Size: 5
    Max size: 6
    000: id: 346, fame: 568, wait_time: 12 (priority: 688)
        001: id: 798, fame: 345, wait_time: 0 (priority: 345)
        002: id: 906, fame: 76, wait_time: 23 (priority: 306)
        003: id: 468, fame: 397, wait_time: 18 (priority: 577)
            004: id: 269, fame: 175, wait_time: 10 (priority: 275)
```

Return values: `-1, 5`

Example #4:

```
queue =
    Size: 5
    Max size: 5
    000: id: 14, fame: 918, wait_time: 11 (priority: 1028)
        001: id: 514, fame: 742, wait_time: 26 (priority: 1002)
        002: id: 852, fame: 644, wait_time: 11 (priority: 754)
        003: id: 235, fame: 416, wait_time: 17 (priority: 586)
            004: id: 194, fame: 601, wait_time: 2 (priority: 621)
customer = id: 931, fame: 62, wait_time: 20 (priority: 262)
```

Updated queue:

```
    Size: 5
    Max size: 5
    000: id: 14, fame: 918, wait_time: 11 (priority: 1028)
        001: id: 514, fame: 742, wait_time: 26 (priority: 1002)
        002: id: 852, fame: 644, wait_time: 11 (priority: 754)
        003: id: 235, fame: 416, wait_time: 17 (priority: 586)
            004: id: 194, fame: 601, wait_time: 2 (priority: 621)
```

Return values: `-1, 5`

Part 6: Heapify-down a Portion of a Priority Queue

`int heapify_down(CustomerQueue* queue, int index)`

This function performs a heapify-down procedure on a `CustomerQueue`, starting at a given index in the queue's `customers` array. The pseudocode below has been adapted adapted from Wikipedia's page on max-heaps.

```
Max-Heapify(A, index):  # A[] represents the customers[] array
    while index < queue.size
        left = 3*index + 1
        mid = 3*index + 2
        right = 3*index + 3
        largest = index

        if left < queue.size and A[left] > A[largest] then
            largest = left

        if mid < queue.size and A[mid] > A[largest] then
            largest = mid

        if right < queue.size and A[right] > A[largest] then
            largest = right

        if largest != index then
            swap A[index] and A[largest]
            index = largest
        else
            break
```

The function takes the following arguments, in this order:

- `queue`: a pointer to a `CustomerQueue` struct. The queue might be empty. Note: for testing purposes, the queue's `customers` array will sometimes consist of a randomly-shuffled array of `Customer` objects. If you implement the above pseudocode correctly, your code will execute properly and terminate cleanly.
- `index`: the 0-based index at which to start the heapify-down procedure

Returns in `$v0`:

- the number of swaps that were performed during the execution of the algorithm; or
- `-1` if `index` <0 or `index` $\geq$ `queue.size`

Additional requirements:

- The function must call `compare_to` and `memcpy`.
- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 10
    Max size: 12
    000: id: 250, fame: 859, wait_time: 16 (priority: 1019)
        001: id: 713, fame: 562, wait_time: 16 (priority: 722)
        002: id: 285, fame: 414, wait_time: 22 (priority: 634)
        003: id: 566, fame: 262, wait_time: 29 (priority: 552)
            004: id: 566, fame: 166, wait_time: 7 (priority: 236)
            005: id: 889, fame: 407, wait_time: 17 (priority: 577)
            006: id: 992, fame: 475, wait_time: 23 (priority: 705)
            007: id: 841, fame: 237, wait_time: 26 (priority: 497)
            008: id: 677, fame: 369, wait_time: 12 (priority: 489)
            009: id: 645, fame: 351, wait_time: 18 (priority: 531)
index = 0
```

Updated queue:

```
    Size: 10
    Max size: 12
    000: id: 250, fame: 859, wait_time: 16 (priority: 1019)
        001: id: 713, fame: 562, wait_time: 16 (priority: 722)
        002: id: 285, fame: 414, wait_time: 22 (priority: 634)
        003: id: 566, fame: 262, wait_time: 29 (priority: 552)
            004: id: 566, fame: 166, wait_time: 7 (priority: 236)
            005: id: 889, fame: 407, wait_time: 17 (priority: 577)
            006: id: 992, fame: 475, wait_time: 23 (priority: 705)
            007: id: 841, fame: 237, wait_time: 26 (priority: 497)
            008: id: 677, fame: 369, wait_time: 12 (priority: 489)
            009: id: 645, fame: 351, wait_time: 18 (priority: 531)
```

Return value: `0`

Example #2:

```
queue =
    Size: 12
    Max size: 15
    000: id: 449, fame: 980, wait_time: 26 (priority: 1240)
        001: id: 835, fame: 967, wait_time: 5 (priority: 1017)
        002: id: 813, fame: 949, wait_time: 27 (priority: 1219)
        003: id: 8, fame: 874, wait_time: 3 (priority: 904)
            004: id: 30, fame: 552, wait_time: 8 (priority: 632)
            005: id: 377, fame: 460, wait_time: 25 (priority: 710)
            006: id: 356, fame: 631, wait_time: 9 (priority: 721)
            007: id: 34, fame: 490, wait_time: 21 (priority: 700)
            008: id: 781, fame: 665, wait_time: 6 (priority: 725)
            009: id: 783, fame: 779, wait_time: 0 (priority: 779)
            010: id: 173, fame: 482, wait_time: 0 (priority: 482)
            011: id: 929, fame: 63, wait_time: 3 (priority: 93)
index = 4
```

Updated queue:

```
    Size: 12
    Max size: 15
    000: id: 449, fame: 980, wait_time: 26 (priority: 1240)
        001: id: 835, fame: 967, wait_time: 5 (priority: 1017)
        002: id: 813, fame: 949, wait_time: 27 (priority: 1219)
        003: id: 8, fame: 874, wait_time: 3 (priority: 904)
            004: id: 30, fame: 552, wait_time: 8 (priority: 632)
            005: id: 377, fame: 460, wait_time: 25 (priority: 710)
            006: id: 356, fame: 631, wait_time: 9 (priority: 721)
            007: id: 34, fame: 490, wait_time: 21 (priority: 700)
            008: id: 781, fame: 665, wait_time: 6 (priority: 725)
            009: id: 783, fame: 779, wait_time: 0 (priority: 779)
            010: id: 173, fame: 482, wait_time: 0 (priority: 482)
            011: id: 929, fame: 63, wait_time: 3 (priority: 93)
```

Return value: `0`

Example #3:

```
queue =
    Size: 9
    Max size: 14
    000: id: 844, fame: 808, wait_time: 26 (priority: 1068)
        001: id: 754, fame: 839, wait_time: 25 (priority: 1089)
        002: id: 607, fame: 111, wait_time: 5 (priority: 161)
        003: id: 563, fame: 969, wait_time: 14 (priority: 1109)
            004: id: 955, fame: 17, wait_time: 21 (priority: 227)
            005: id: 445, fame: 955, wait_time: 13 (priority: 1085)
            006: id: 934, fame: 206, wait_time: 30 (priority: 506)
            007: id: 379, fame: 115, wait_time: 15 (priority: 265)
            008: id: 926, fame: 994, wait_time: 17 (priority: 1164)
index = 0
```

Updated queue:

```
    Size: 9
    Max size: 14
    000: id: 563, fame: 969, wait_time: 14 (priority: 1109)
        001: id: 754, fame: 839, wait_time: 25 (priority: 1089)
        002: id: 607, fame: 111, wait_time: 5 (priority: 161)
        003: id: 844, fame: 808, wait_time: 26 (priority: 1068)
            004: id: 955, fame: 17, wait_time: 21 (priority: 227)
            005: id: 445, fame: 955, wait_time: 13 (priority: 1085)
            006: id: 934, fame: 206, wait_time: 30 (priority: 506)
            007: id: 379, fame: 115, wait_time: 15 (priority: 265)
            008: id: 926, fame: 994, wait_time: 17 (priority: 1164)
```

Return value: `1`

Example #4:

```
queue =
    Size: 11
    Max size: 17
    000: id: 797, fame: 183, wait_time: 18 (priority: 363)
        001: id: 367, fame: 462, wait_time: 15 (priority: 612)
        002: id: 121, fame: 532, wait_time: 24 (priority: 772)
        003: id: 451, fame: 668, wait_time: 0 (priority: 668)
            004: id: 445, fame: 850, wait_time: 17 (priority: 1020)
            005: id: 805, fame: 42, wait_time: 15 (priority: 192)
            006: id: 955, fame: 525, wait_time: 20 (priority: 725)
            007: id: 869, fame: 553, wait_time: 3 (priority: 583)
            008: id: 537, fame: 829, wait_time: 26 (priority: 1089)
            009: id: 315, fame: 948, wait_time: 17 (priority: 1118)
            010: id: 851, fame: 472, wait_time: 26 (priority: 732)
index = 0
```

Updated queue:

```
    Size: 11
    Max size: 17
    000: id: 121, fame: 532, wait_time: 24 (priority: 772)
        001: id: 367, fame: 462, wait_time: 15 (priority: 612)
        002: id: 315, fame: 948, wait_time: 17 (priority: 1118)
        003: id: 451, fame: 668, wait_time: 0 (priority: 668)
            004: id: 445, fame: 850, wait_time: 17 (priority: 1020)
            005: id: 805, fame: 42, wait_time: 15 (priority: 192)
            006: id: 955, fame: 525, wait_time: 20 (priority: 725)
            007: id: 869, fame: 553, wait_time: 3 (priority: 583)
            008: id: 537, fame: 829, wait_time: 26 (priority: 1089)
            009: id: 797, fame: 183, wait_time: 18 (priority: 363)
            010: id: 851, fame: 472, wait_time: 26 (priority: 732)
```

Return value: `2`

Example #5:

```
queue =
    Size: 50
    Max size: 60
    000: id: 703, fame: 29, wait_time: 10 (priority: 129)
        001: id: 431, fame: 542, wait_time: 2 (priority: 562)
        002: id: 901, fame: 600, wait_time: 27 (priority: 870)
        003: id: 595, fame: 419, wait_time: 17 (priority: 589)
            004: id: 917, fame: 879, wait_time: 8 (priority: 959)
            005: id: 231, fame: 928, wait_time: 23 (priority: 1158)
            006: id: 481, fame: 601, wait_time: 15 (priority: 751)
            007: id: 219, fame: 831, wait_time: 17 (priority: 1001)
            008: id: 55, fame: 331, wait_time: 14 (priority: 471)
            009: id: 316, fame: 549, wait_time: 21 (priority: 759)
            010: id: 610, fame: 454, wait_time: 6 (priority: 514)
            011: id: 565, fame: 361, wait_time: 28 (priority: 641)
            012: id: 732, fame: 302, wait_time: 8 (priority: 382)
                013: id: 493, fame: 799, wait_time: 4 (priority: 839)
                014: id: 564, fame: 656, wait_time: 28 (priority: 936)
                015: id: 433, fame: 907, wait_time: 25 (priority: 1157)
                016: id: 183, fame: 116, wait_time: 9 (priority: 206)
                017: id: 476, fame: 899, wait_time: 30 (priority: 1199)
                018: id: 756, fame: 12, wait_time: 21 (priority: 222)
                019: id: 677, fame: 161, wait_time: 21 (priority: 371)
                020: id: 432, fame: 744, wait_time: 2 (priority: 764)
                021: id: 571, fame: 520, wait_time: 16 (priority: 680)
                022: id: 772, fame: 983, wait_time: 30 (priority: 1283)
                023: id: 291, fame: 593, wait_time: 22 (priority: 813)
                024: id: 608, fame: 685, wait_time: 14 (priority: 825)
                025: id: 746, fame: 458, wait_time: 15 (priority: 608)
                026: id: 391, fame: 18, wait_time: 20 (priority: 218)
                027: id: 795, fame: 705, wait_time: 8 (priority: 785)
                028: id: 144, fame: 802, wait_time: 21 (priority: 1012)
                029: id: 998, fame: 796, wait_time: 12 (priority: 916)
                030: id: 413, fame: 926, wait_time: 27 (priority: 1196)
                031: id: 656, fame: 396, wait_time: 10 (priority: 496)
                032: id: 328, fame: 412, wait_time: 18 (priority: 592)
                033: id: 670, fame: 998, wait_time: 21 (priority: 1208)
                034: id: 338, fame: 566, wait_time: 14 (priority: 706)
                035: id: 419, fame: 524, wait_time: 23 (priority: 754)
                036: id: 607, fame: 333, wait_time: 15 (priority: 483)
                037: id: 621, fame: 128, wait_time: 0 (priority: 128)
                038: id: 964, fame: 135, wait_time: 12 (priority: 255)
                039: id: 889, fame: 971, wait_time: 6 (priority: 1031)
                    040: id: 828, fame: 423, wait_time: 8 (priority: 503)
                    041: id: 55, fame: 401, wait_time: 24 (priority: 641)
                    042: id: 575, fame: 337, wait_time: 26 (priority: 597)
                    043: id: 51, fame: 542, wait_time: 8 (priority: 622)
                    044: id: 34, fame: 953, wait_time: 10 (priority: 1053)
                    045: id: 774, fame: 950, wait_time: 0 (priority: 950)
                    046: id: 297, fame: 977, wait_time: 14 (priority: 1117)
                    047: id: 75, fame: 614, wait_time: 15 (priority: 764)
                    048: id: 746, fame: 548, wait_time: 28 (priority: 828)
                    049: id: 803, fame: 566, wait_time: 4 (priority: 606)
index = 1
```

Updated queue:

```
    Size: 50
    Max size: 60
    000: id: 703, fame: 29, wait_time: 10 (priority: 129)
        001: id: 231, fame: 928, wait_time: 23 (priority: 1158)
        002: id: 901, fame: 600, wait_time: 27 (priority: 870)
        003: id: 595, fame: 419, wait_time: 17 (priority: 589)
```

```
004: id: 917, fame: 879, wait_time: 8 (priority: 959)
005: id: 476, fame: 899, wait_time: 30 (priority: 1199)
006: id: 481, fame: 601, wait_time: 15 (priority: 751)
007: id: 219, fame: 831, wait_time: 17 (priority: 1001)
008: id: 55, fame: 331, wait_time: 14 (priority: 471)
009: id: 316, fame: 549, wait_time: 21 (priority: 759)
010: id: 610, fame: 454, wait_time: 6 (priority: 514)
011: id: 565, fame: 361, wait_time: 28 (priority: 641)
012: id: 732, fame: 302, wait_time: 8 (priority: 382)
    013: id: 493, fame: 799, wait_time: 4 (priority: 839)
    014: id: 564, fame: 656, wait_time: 28 (priority: 936)
    015: id: 433, fame: 907, wait_time: 25 (priority: 1157)
    016: id: 183, fame: 116, wait_time: 9 (priority: 206)
    017: id: 431, fame: 542, wait_time: 2 (priority: 562)
    018: id: 756, fame: 12, wait_time: 21 (priority: 222)
    019: id: 677, fame: 161, wait_time: 21 (priority: 371)
    020: id: 432, fame: 744, wait_time: 2 (priority: 764)
    021: id: 571, fame: 520, wait_time: 16 (priority: 680)
    022: id: 772, fame: 983, wait_time: 30 (priority: 1283)
    023: id: 291, fame: 593, wait_time: 22 (priority: 813)
    024: id: 608, fame: 685, wait_time: 14 (priority: 825)
    025: id: 746, fame: 458, wait_time: 15 (priority: 608)
    026: id: 391, fame: 18, wait_time: 20 (priority: 218)
    027: id: 795, fame: 705, wait_time: 8 (priority: 785)
    028: id: 144, fame: 802, wait_time: 21 (priority: 1012)
    029: id: 998, fame: 796, wait_time: 12 (priority: 916)
    030: id: 413, fame: 926, wait_time: 27 (priority: 1196)
    031: id: 656, fame: 396, wait_time: 10 (priority: 496)
    032: id: 328, fame: 412, wait_time: 18 (priority: 592)
    033: id: 670, fame: 998, wait_time: 21 (priority: 1208)
    034: id: 338, fame: 566, wait_time: 14 (priority: 706)
    035: id: 419, fame: 524, wait_time: 23 (priority: 754)
    036: id: 607, fame: 333, wait_time: 15 (priority: 483)
    037: id: 621, fame: 128, wait_time: 0 (priority: 128)
    038: id: 964, fame: 135, wait_time: 12 (priority: 255)
    039: id: 889, fame: 971, wait_time: 6 (priority: 1031)
        040: id: 828, fame: 423, wait_time: 8 (priority: 503)
        041: id: 55, fame: 401, wait_time: 24 (priority: 641)
        042: id: 575, fame: 337, wait_time: 26 (priority: 597)
        043: id: 51, fame: 542, wait_time: 8 (priority: 622)
        044: id: 34, fame: 953, wait_time: 10 (priority: 1053)
        045: id: 774, fame: 950, wait_time: 0 (priority: 950)
        046: id: 297, fame: 977, wait_time: 14 (priority: 1117)
        047: id: 75, fame: 614, wait_time: 15 (priority: 764)
        048: id: 746, fame: 548, wait_time: 28 (priority: 828)
        049: id: 803, fame: 566, wait_time: 4 (priority: 606)
```

Return value: `2`

Part 7: Dequeue the First Customer from the Queue

`int dequeue(CustomerQueue* queue, Customer* dequeued_customer)`

This function dequeues the highest-priority item from the queue (which is always at `customers[0]`) and copies it into `dequeued_customer`. As is done in the standard dequeue algorithm, the element at `queue.customers[queue.size-1]` is then copied to `queue.customers[0]`. (Do not zero-out the element at `queue.customers[queue.size-1]`.) Next, the value of `queue.size` is decremented by 1. Finally, the function calls `heapify_down` on index 0 of the `queue.customers` array to re-heapify the queue's array.

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. The queue might be empty or full.
- `dequeued_customer`: a pointer to an uninitialized `Customer` struct. Assume the struct is full of garbage.

Returns in `$v0`:

- the size of the queue after removing the highest-priority item, or
- `-1` if the queue is empty

Additional requirements:

- The function must call `memcpy` and `heapify_down`.
- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 5
    Max size: 6
    000: id: 111, fame: 859, wait_time: 28 (priority: 1139)
        001: id: 550, fame: 576, wait_time: 10 (priority: 676)
        002: id: 788, fame: 418, wait_time: 8 (priority: 498)
        003: id: 896, fame: 40, wait_time: 12 (priority: 160)
            004: id: 54, fame: 275, wait_time: 2 (priority: 295)
```

Return value: `4`

Resulting `dequeued_customer_expected`: `id: 111, fame: 859, wait_time: 28 (priority: 1139)`

Updated queue:

```
    Size: 4
    Max size: 6
    000: id: 550, fame: 576, wait_time: 10 (priority: 676)
        001: id: 54, fame: 275, wait_time: 2 (priority: 295)
        002: id: 788, fame: 418, wait_time: 8 (priority: 498)
        003: id: 896, fame: 40, wait_time: 12 (priority: 160)
```

Example #2:

```
queue =
    Size: 29
    Max size: 30
    000: id: 579, fame: 829, wait_time: 30 (priority: 1129)
        001: id: 130, fame: 810, wait_time: 29 (priority: 1100)
        002: id: 90, fame: 900, wait_time: 19 (priority: 1090)
        003: id: 272, fame: 481, wait_time: 26 (priority: 741)
            004: id: 38, fame: 788, wait_time: 9 (priority: 878)
            005: id: 324, fame: 750, wait_time: 14 (priority: 890)
            006: id: 915, fame: 788, wait_time: 20 (priority: 988)
            007: id: 683, fame: 611, wait_time: 28 (priority: 891)
            008: id: 744, fame: 840, wait_time: 2 (priority: 860)
            009: id: 105, fame: 928, wait_time: 1 (priority: 938)
            010: id: 575, fame: 342, wait_time: 22 (priority: 562)
            011: id: 22, fame: 299, wait_time: 29 (priority: 589)
            012: id: 441, fame: 588, wait_time: 15 (priority: 738)
                013: id: 164, fame: 139, wait_time: 10 (priority: 239)
                014: id: 352, fame: 535, wait_time: 25 (priority: 785)
                015: id: 495, fame: 212, wait_time: 26 (priority: 472)
                016: id: 387, fame: 72, wait_time: 0 (priority: 72)
                017: id: 624, fame: 655, wait_time: 16 (priority: 815)
                018: id: 323, fame: 13, wait_time: 12 (priority: 133)
                019: id: 928, fame: 447, wait_time: 24 (priority: 687)
                020: id: 526, fame: 445, wait_time: 21 (priority: 655)
                021: id: 552, fame: 654, wait_time: 25 (priority: 904)
                022: id: 252, fame: 168, wait_time: 0 (priority: 168)
                023: id: 854, fame: 498, wait_time: 16 (priority: 658)
                024: id: 685, fame: 430, wait_time: 11 (priority: 540)
                025: id: 523, fame: 33, wait_time: 27 (priority: 303)
                026: id: 773, fame: 661, wait_time: 5 (priority: 711)
                027: id: 95, fame: 684, wait_time: 4 (priority: 724)
                028: id: 936, fame: 499, wait_time: 21 (priority: 709)
```

Return value: `28`

Resulting `dequeued_customer_expected`: `id: 579, fame: 829, wait_time: 30 (priority: 1129)`

Updated queue:

```
    Size: 28
    Max size: 30
    000: id: 130, fame: 810, wait_time: 29 (priority: 1100)
        001: id: 915, fame: 788, wait_time: 20 (priority: 988)
        002: id: 90, fame: 900, wait_time: 19 (priority: 1090)
        003: id: 272, fame: 481, wait_time: 26 (priority: 741)
            004: id: 38, fame: 788, wait_time: 9 (priority: 878)
            005: id: 324, fame: 750, wait_time: 14 (priority: 890)
            006: id: 552, fame: 654, wait_time: 25 (priority: 904)
            007: id: 683, fame: 611, wait_time: 28 (priority: 891)
            008: id: 744, fame: 840, wait_time: 2 (priority: 860)
            009: id: 105, fame: 928, wait_time: 1 (priority: 938)
            010: id: 575, fame: 342, wait_time: 22 (priority: 562)
            011: id: 22, fame: 299, wait_time: 29 (priority: 589)
            012: id: 441, fame: 588, wait_time: 15 (priority: 738)
                013: id: 164, fame: 139, wait_time: 10 (priority: 239)
                014: id: 352, fame: 535, wait_time: 25 (priority: 785)
                015: id: 495, fame: 212, wait_time: 26 (priority: 472)
                016: id: 387, fame: 72, wait_time: 0 (priority: 72)
                017: id: 624, fame: 655, wait_time: 16 (priority: 815)
                018: id: 323, fame: 13, wait_time: 12 (priority: 133)
                019: id: 928, fame: 447, wait_time: 24 (priority: 687)
                020: id: 526, fame: 445, wait_time: 21 (priority: 655)
                021: id: 936, fame: 499, wait_time: 21 (priority: 709)
                022: id: 252, fame: 168, wait_time: 0 (priority: 168)
                023: id: 854, fame: 498, wait_time: 16 (priority: 658)
                024: id: 685, fame: 430, wait_time: 11 (priority: 540)
                025: id: 523, fame: 33, wait_time: 27 (priority: 303)
                026: id: 773, fame: 661, wait_time: 5 (priority: 711)
                027: id: 95, fame: 684, wait_time: 4 (priority: 724)
```

Example #3:

```
queue =
    Size: 0
    Max size: 5
```

Return value: `-1`

Resulting `dequeued_customer_expected`: original garbage values

Updated queue:

```
    Size: 0
    Max size: 5
```

Part 8: Build a Heap from an Unsorted Array of Customer Structs

`int build_heap(CustomerQueue* queue)`

This function takes an unsorted array of `Customer` objects, stored in a `CustomerQueue`'s `customers` array, and reorganizes the elements to transform the `customers` array into a ternary max-heap. You may assume that the `size` attribute of the queue is size of the `customers` array.

The pseudocode below to implemented is adapted from the Build-Max-Heap pseudocode on...Wikipedia!

```
if queue.size == 0 then
    return 0
res = 0
index = (queue.size - 1) / 3  # integer division
```

```
for i in index downto 0
    res += heapify_down(queue, i)
return res
```

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. The queue might be empty.

Returns in `$v0`:

- the sum of return values returned by the series of calls to `heapify_down`

Additional requirements:

- The function must call `heapify_down`.
- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 15
    Max size: 18
    000: id: 103, fame: 1, wait_time: 27 (priority: 271)
        001: id: 317, fame: 258, wait_time: 13 (priority: 388)
        002: id: 815, fame: 730, wait_time: 24 (priority: 970)
        003: id: 887, fame: 182, wait_time: 25 (priority: 432)
            004: id: 365, fame: 889, wait_time: 11 (priority: 999)
            005: id: 994, fame: 104, wait_time: 26 (priority: 364)
            006: id: 667, fame: 330, wait_time: 8 (priority: 410)
            007: id: 318, fame: 565, wait_time: 4 (priority: 605)
            008: id: 38, fame: 431, wait_time: 23 (priority: 661)
            009: id: 179, fame: 346, wait_time: 21 (priority: 556)
            010: id: 135, fame: 81, wait_time: 27 (priority: 351)
            011: id: 400, fame: 327, wait_time: 25 (priority: 577)
            012: id: 281, fame: 749, wait_time: 15 (priority: 899)
                013: id: 5, fame: 181, wait_time: 2 (priority: 201)
                014: id: 519, fame: 346, wait_time: 21 (priority: 556)
```

Return value: `6`

Updated queue:

```
    Size: 15
    Max size: 18
    000: id: 365, fame: 889, wait_time: 11 (priority: 999)
        001: id: 519, fame: 346, wait_time: 21 (priority: 556)
        002: id: 815, fame: 730, wait_time: 24 (priority: 970)
        003: id: 281, fame: 749, wait_time: 15 (priority: 899)
            004: id: 317, fame: 258, wait_time: 13 (priority: 388)
            005: id: 994, fame: 104, wait_time: 26 (priority: 364)
            006: id: 667, fame: 330, wait_time: 8 (priority: 410)
            007: id: 318, fame: 565, wait_time: 4 (priority: 605)
            008: id: 38, fame: 431, wait_time: 23 (priority: 661)
            009: id: 179, fame: 346, wait_time: 21 (priority: 556)
            010: id: 135, fame: 81, wait_time: 27 (priority: 351)
            011: id: 400, fame: 327, wait_time: 25 (priority: 577)
            012: id: 887, fame: 182, wait_time: 25 (priority: 432)
                013: id: 5, fame: 181, wait_time: 2 (priority: 201)
                014: id: 103, fame: 1, wait_time: 27 (priority: 271)
```

Example #2:

```
queue =
    Size: 25
    Max size: 35
    000: id: 63, fame: 622, wait_time: 17 (priority: 792)
        001: id: 559, fame: 48, wait_time: 1 (priority: 58)
        002: id: 598, fame: 124, wait_time: 26 (priority: 384)
        003: id: 635, fame: 859, wait_time: 29 (priority: 1149)
            004: id: 500, fame: 844, wait_time: 24 (priority: 1084)
            005: id: 154, fame: 779, wait_time: 29 (priority: 1069)
            006: id: 548, fame: 84, wait_time: 2 (priority: 104)
            007: id: 140, fame: 525, wait_time: 13 (priority: 655)
            008: id: 639, fame: 402, wait_time: 24 (priority: 642)
            009: id: 534, fame: 691, wait_time: 17 (priority: 861)
            010: id: 996, fame: 354, wait_time: 4 (priority: 394)
            011: id: 876, fame: 43, wait_time: 22 (priority: 263)
            012: id: 61, fame: 384, wait_time: 5 (priority: 434)
                013: id: 436, fame: 578, wait_time: 27 (priority: 848)
                014: id: 949, fame: 914, wait_time: 11 (priority: 1024)
                015: id: 558, fame: 397, wait_time: 20 (priority: 597)
                016: id: 712, fame: 331, wait_time: 28 (priority: 611)
                017: id: 76, fame: 97, wait_time: 15 (priority: 247)
                018: id: 939, fame: 331, wait_time: 8 (priority: 411)
                019: id: 475, fame: 545, wait_time: 19 (priority: 735)
                020: id: 649, fame: 645, wait_time: 21 (priority: 855)
                021: id: 601, fame: 564, wait_time: 14 (priority: 704)
                022: id: 312, fame: 579, wait_time: 30 (priority: 879)
                023: id: 865, fame: 680, wait_time: 30 (priority: 980)
                024: id: 726, fame: 743, wait_time: 4 (priority: 783)
```

Return value: `7`

Updated queue:

```
    Size: 25
    Max size: 35
    000: id: 635, fame: 859, wait_time: 29 (priority: 1149)
        001: id: 500, fame: 844, wait_time: 24 (priority: 1084)
        002: id: 865, fame: 680, wait_time: 30 (priority: 980)
        003: id: 63, fame: 622, wait_time: 17 (priority: 792)
            004: id: 949, fame: 914, wait_time: 11 (priority: 1024)
```

```
005: id: 154, fame: 779, wait_time: 29 (priority: 1069)
006: id: 649, fame: 645, wait_time: 21 (priority: 855)
007: id: 312, fame: 579, wait_time: 30 (priority: 879)
008: id: 639, fame: 402, wait_time: 24 (priority: 642)
009: id: 534, fame: 691, wait_time: 17 (priority: 861)
010: id: 996, fame: 354, wait_time: 4 (priority: 394)
011: id: 876, fame: 43, wait_time: 22 (priority: 263)
012: id: 61, fame: 384, wait_time: 5 (priority: 434)
    013: id: 436, fame: 578, wait_time: 27 (priority: 848)
    014: id: 559, fame: 48, wait_time: 1 (priority: 58)
    015: id: 558, fame: 397, wait_time: 20 (priority: 597)
    016: id: 712, fame: 331, wait_time: 28 (priority: 611)
    017: id: 76, fame: 97, wait_time: 15 (priority: 247)
    018: id: 939, fame: 331, wait_time: 8 (priority: 411)
    019: id: 475, fame: 545, wait_time: 19 (priority: 735)
    020: id: 548, fame: 84, wait_time: 2 (priority: 104)
    021: id: 601, fame: 564, wait_time: 14 (priority: 704)
    022: id: 598, fame: 124, wait_time: 26 (priority: 384)
    023: id: 140, fame: 525, wait_time: 13 (priority: 655)
    024: id: 726, fame: 743, wait_time: 4 (priority: 783)
```

Part 9: Increment the Waiting Time of Customers in a Queue

`int increment_time(CustomerQueue* queue, int delta_mins, int fame_level)`

The restaurant has become so popular, that it has become *the* go-to place for the rich and beautiful. In fact, just by virtue of standing in line do normal folks become famous, thanks to the aggressive paparazzi who stalk the waiting line.

This function increments the `wait_time` field of every customer waiting on line to get into the restaurant, as represented by the `queue.customers` array. Moreover, every customer whose fame is less than `fame_level` gets a boost to his `fame` attribute equal to `delta_mins`. Consequently, less-famous people could be made more famous than others ahead them on line! Therefore, after updating the `wait_time` for every customer and the `fame` for the appropriate customers, call `build_heap` to re-heapify the priority queue.

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. The queue might be empty.
- `delta_mins`: a positive integer
- `fame_level`: a positive integer

Returns in `$v0`:

- the average waiting time of the customers in the queue, after incrementing every customer's `wait_time` by `delta_mins` (use integer division); or
- `-1` if the queue is empty, `delta_mins` ≤ 0 or `fame_level` ≤ 0

Additional requirements:

- The function must call `build_heap`.

- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 10
    Max size: 15
    000: id: 606, fame: 89, wait_time: 24 (priority: 329)
        001: id: 419, fame: 90, wait_time: 17 (priority: 260)
        002: id: 347, fame: 80, wait_time: 9 (priority: 170)
        003: id: 120, fame: 13, wait_time: 0 (priority: 13)
            004: id: 883, fame: 49, wait_time: 5 (priority: 99)
            005: id: 311, fame: 49, wait_time: 20 (priority: 249)
            006: id: 161, fame: 89, wait_time: 16 (priority: 249)
            007: id: 231, fame: 29, wait_time: 0 (priority: 29)
            008: id: 687, fame: 10, wait_time: 11 (priority: 120)
            009: id: 163, fame: 9, wait_time: 16 (priority: 169)
delta_mins = 30
fame_level = 50
```

Return value: `41`

Updated queue:

```
    Size: 10
    Max size: 15
    000: id: 606, fame: 89, wait_time: 54 (priority: 629)
        001: id: 311, fame: 79, wait_time: 50 (priority: 579)
        002: id: 163, fame: 39, wait_time: 46 (priority: 499)
        003: id: 120, fame: 43, wait_time: 30 (priority: 343)
            004: id: 883, fame: 79, wait_time: 35 (priority: 429)
            005: id: 419, fame: 90, wait_time: 47 (priority: 560)
            006: id: 161, fame: 89, wait_time: 46 (priority: 549)
            007: id: 231, fame: 59, wait_time: 30 (priority: 359)
            008: id: 687, fame: 40, wait_time: 41 (priority: 450)
            009: id: 347, fame: 80, wait_time: 39 (priority: 470)
```

Example #2:

```
queue =
    Size: 18
    Max size: 20
    000: id: 632, fame: 95, wait_time: 26 (priority: 355)
        001: id: 690, fame: 88, wait_time: 26 (priority: 348)
        002: id: 731, fame: 84, wait_time: 23 (priority: 314)
        003: id: 814, fame: 45, wait_time: 21 (priority: 255)
            004: id: 489, fame: 25, wait_time: 29 (priority: 315)
            005: id: 557, fame: 46, wait_time: 30 (priority: 346)
            006: id: 194, fame: 96, wait_time: 8 (priority: 176)
            007: id: 447, fame: 95, wait_time: 1 (priority: 105)
            008: id: 36, fame: 13, wait_time: 12 (priority: 133)
            009: id: 831, fame: 61, wait_time: 11 (priority: 171)
            010: id: 111, fame: 38, wait_time: 20 (priority: 238)
            011: id: 155, fame: 59, wait_time: 15 (priority: 209)
            012: id: 646, fame: 70, wait_time: 18 (priority: 250)
                013: id: 603, fame: 41, wait_time: 8 (priority: 121)
                014: id: 45, fame: 2, wait_time: 5 (priority: 52)
                015: id: 229, fame: 75, wait_time: 0 (priority: 75)
                016: id: 254, fame: 6, wait_time: 1 (priority: 16)
                017: id: 772, fame: 72, wait_time: 20 (priority: 272)
delta_mins = 20
fame_level = 50
```

Return value: `35`

Updated queue:

```
    Size: 18
    Max size: 20
    000: id: 557, fame: 66, wait_time: 50 (priority: 566)
        001: id: 632, fame: 95, wait_time: 46 (priority: 555)
        002: id: 731, fame: 84, wait_time: 43 (priority: 514)
        003: id: 814, fame: 65, wait_time: 41 (priority: 475)
            004: id: 489, fame: 45, wait_time: 49 (priority: 535)
            005: id: 690, fame: 88, wait_time: 46 (priority: 548)
            006: id: 194, fame: 96, wait_time: 28 (priority: 376)
            007: id: 447, fame: 95, wait_time: 21 (priority: 305)
            008: id: 36, fame: 33, wait_time: 32 (priority: 353)
            009: id: 831, fame: 61, wait_time: 31 (priority: 371)
            010: id: 111, fame: 58, wait_time: 40 (priority: 458)
            011: id: 155, fame: 59, wait_time: 35 (priority: 409)
            012: id: 646, fame: 70, wait_time: 38 (priority: 450)
                013: id: 603, fame: 61, wait_time: 28 (priority: 341)
                014: id: 45, fame: 22, wait_time: 25 (priority: 272)
                015: id: 229, fame: 75, wait_time: 20 (priority: 275)
                016: id: 254, fame: 26, wait_time: 21 (priority: 236)
                017: id: 772, fame: 72, wait_time: 40 (priority: 472)
```

Example #3:

```
queue =
    Size: 18
    Max size: 20
    000: id: 287, fame: 34, wait_time: 30 (priority: 334)
        001: id: 886, fame: 43, wait_time: 19 (priority: 233)
        002: id: 537, fame: 25, wait_time: 21 (priority: 235)
        003: id: 779, fame: 80, wait_time: 21 (priority: 290)
            004: id: 955, fame: 1, wait_time: 23 (priority: 231)
            005: id: 413, fame: 58, wait_time: 17 (priority: 228)
            006: id: 635, fame: 83, wait_time: 9 (priority: 173)
            007: id: 65, fame: 1, wait_time: 2 (priority: 21)
            008: id: 622, fame: 56, wait_time: 1 (priority: 66)
            009: id: 446, fame: 89, wait_time: 5 (priority: 139)
            010: id: 578, fame: 15, wait_time: 12 (priority: 135)
            011: id: 879, fame: 8, wait_time: 26 (priority: 268)
            012: id: 997, fame: 97, wait_time: 16 (priority: 257)
                013: id: 370, fame: 29, wait_time: 3 (priority: 59)
                014: id: 560, fame: 9, wait_time: 15 (priority: 159)
                015: id: 291, fame: 52, wait_time: 9 (priority: 142)
                016: id: 486, fame: 23, wait_time: 4 (priority: 63)
                017: id: 75, fame: 17, wait_time: 8 (priority: 97)
delta_mins = 15
fame_level = 25
```

Return value: `28`

Updated queue:

```
    Size: 18
    Max size: 20
    000: id: 287, fame: 34, wait_time: 45 (priority: 484)
        001: id: 955, fame: 16, wait_time: 38 (priority: 396)
        002: id: 537, fame: 25, wait_time: 36 (priority: 385)
        003: id: 779, fame: 80, wait_time: 36 (priority: 440)
            004: id: 886, fame: 43, wait_time: 34 (priority: 383)
            005: id: 413, fame: 58, wait_time: 32 (priority: 378)
            006: id: 635, fame: 83, wait_time: 24 (priority: 323)
            007: id: 65, fame: 16, wait_time: 17 (priority: 186)
            008: id: 622, fame: 56, wait_time: 16 (priority: 216)
            009: id: 446, fame: 89, wait_time: 20 (priority: 289)
            010: id: 578, fame: 30, wait_time: 27 (priority: 300)
            011: id: 879, fame: 23, wait_time: 41 (priority: 433)
            012: id: 997, fame: 97, wait_time: 31 (priority: 407)
                013: id: 370, fame: 29, wait_time: 18 (priority: 209)
                014: id: 560, fame: 24, wait_time: 30 (priority: 324)
                015: id: 291, fame: 52, wait_time: 24 (priority: 292)
                016: id: 486, fame: 38, wait_time: 19 (priority: 228)
                017: id: 75, fame: 32, wait_time: 23 (priority: 262)
```

Part 10: Admit Customers into the Restaurant

```
int admit_customers(CustomerQueue* queue, int max_admits, Customer* admitted)
```

This function dequeues at most `max_admits` customers from the given queue, writing them sequentially into the `admitted` array. In this way, the function copies the first `max_admits` 8-byte `Customer` structs from `queue.customers` to `admitted`, *not* pointers to structs. You may assume that the `admitted` array is large enough to store at least `max_admits` `Customer` structs. If all goes well, the customers in the `admitted` array will be sorted in descending order by priority, as determined by `compare_to`. The function need not call `compare_to` directly, of course.

The function takes the following arguments, in this order:

- `queue`: a pointer to a valid `CustomerQueue` struct. The queue might be empty.

- `max_admits`: the maximum number of customers to dequeue from the queue
- `admitted`: a pointer to an uninitialized array of `Customer` structs. Assume the allocated memory is full of garbage.

Returns in `$v0`:

- the number of customers admitted to the restaurant, or
- `-1` if the queue is empty or `max_admits` ≤ 0

Additional requirements:

- The function must call `dequeue`.
- The function must not write any changes to main memory except as necessary.

Example #1:

```
queue =
    Size: 4
    Max size: 6
    000: id: 28, fame: 909, wait_time: 20 (priority: 1109)
        001: id: 642, fame: 611, wait_time: 22 (priority: 831)
        002: id: 905, fame: 154, wait_time: 0 (priority: 154)
        003: id: 855, fame: 652, wait_time: 26 (priority: 912)
max_admits = 3
admitted =  # garbage values
    000: id: 492, fame: 281, wait_time: 13 (priority: 411)
    001: id: 284, fame: 973, wait_time: 12 (priority: 1093)
    002: id: 421, fame: 879, wait_time: 25 (priority: 1129)
    003: id: 205, fame: 417, wait_time: 12 (priority: 537)
    004: id: 667, fame: 408, wait_time: 22 (priority: 628)
```

Return value: `3`

Updated queue:

```
    Size: 1
    Max size: 6
    000: id: 905, fame: 154, wait_time: 0 (priority: 154)
```

Updated admitted:

```
    000: id: 28, fame: 909, wait_time: 20 (priority: 1109)
    001: id: 855, fame: 652, wait_time: 26 (priority: 912)
    002: id: 642, fame: 611, wait_time: 22 (priority: 831)
    003: id: 205, fame: 417, wait_time: 12 (priority: 537)
    004: id: 667, fame: 408, wait_time: 22 (priority: 628)
```

Example #2:

```
queue =
    Size: 9
    Max size: 12
    000: id: 780, fame: 999, wait_time: 29 (priority: 1289)
        001: id: 158, fame: 988, wait_time: 19 (priority: 1178)
        002: id: 357, fame: 740, wait_time: 17 (priority: 910)
        003: id: 497, fame: 381, wait_time: 7 (priority: 451)
            004: id: 944, fame: 413, wait_time: 2 (priority: 433)
            005: id: 472, fame: 928, wait_time: 11 (priority: 1038)
            006: id: 956, fame: 16, wait_time: 24 (priority: 256)
            007: id: 548, fame: 469, wait_time: 14 (priority: 609)
            008: id: 970, fame: 143, wait_time: 4 (priority: 183)
max_admits = 7
admitted =  # garbage values
    000: id: 511, fame: 341, wait_time: 23 (priority: 571)
    001: id: 392, fame: 605, wait_time: 25 (priority: 855)
    002: id: 503, fame: 483, wait_time: 24 (priority: 723)
    003: id: 270, fame: 758, wait_time: 17 (priority: 928)
    004: id: 713, fame: 663, wait_time: 27 (priority: 933)
    005: id: 450, fame: 295, wait_time: 8 (priority: 375)
    006: id: 90, fame: 430, wait_time: 19 (priority: 620)
    007: id: 820, fame: 494, wait_time: 15 (priority: 644)
    008: id: 969, fame: 661, wait_time: 28 (priority: 941)
```

Return value: `7`

Updated queue:

```
    Size: 2
    Max size: 12
    000: id: 956, fame: 16, wait_time: 24 (priority: 256)
        001: id: 970, fame: 143, wait_time: 4 (priority: 183)
```

Updated admitted:

```
    000: id: 780, fame: 999, wait_time: 29 (priority: 1289)
    001: id: 158, fame: 988, wait_time: 19 (priority: 1178)
    002: id: 472, fame: 928, wait_time: 11 (priority: 1038)
    003: id: 357, fame: 740, wait_time: 17 (priority: 910)
    004: id: 548, fame: 469, wait_time: 14 (priority: 609)
    005: id: 497, fame: 381, wait_time: 7 (priority: 451)
    006: id: 944, fame: 413, wait_time: 2 (priority: 433)
    007: id: 820, fame: 494, wait_time: 15 (priority: 644)
    008: id: 969, fame: 661, wait_time: 28 (priority: 941)
```

Example #3:

```
queue =
    Size: 5
    Max size: 8
    000: id: 696, fame: 948, wait_time: 6 (priority: 1008)
        001: id: 703, fame: 674, wait_time: 2 (priority: 694)
        002: id: 855, fame: 354, wait_time: 25 (priority: 604)
        003: id: 902, fame: 321, wait_time: 2 (priority: 341)
```

```
            004: id: 992, fame: 492, wait_time: 3 (priority: 522)
max_admits = 10
admitted =  # garbage values
    000: id: 63, fame: 920, wait_time: 15 (priority: 1070)
    001: id: 486, fame: 82, wait_time: 5 (priority: 132)
    002: id: 484, fame: 999, wait_time: 27 (priority: 1269)
    003: id: 368, fame: 542, wait_time: 19 (priority: 732)
    004: id: 331, fame: 282, wait_time: 11 (priority: 392)
```

Return value: `5`

Updated queue:

```
    Size: 0
    Max size: 8
```

Updated admitted:

```
    000: id: 696, fame: 948, wait_time: 6 (priority: 1008)
    001: id: 703, fame: 674, wait_time: 2 (priority: 694)
    002: id: 855, fame: 354, wait_time: 25 (priority: 604)
    003: id: 992, fame: 492, wait_time: 3 (priority: 522)
    004: id: 902, fame: 321, wait_time: 2 (priority: 341)
```

Part 11: Seat Customers Admitted to the Restaurant

`int seat_customers(Customer* admitted, int num_admitted, int budget)`

This function solves a special instance of the knapsack problem for our restaurant scenario. Customers have finally been admitted into the restaurant. These `num_admitted` customers are stored in the `admitted` array in descending order by priority as 8-byte structs. But...uh oh, there might not be enough "entertainment value" in the restaurant to accommodate everyone's entertainment needs. We can only seat people if we are able to spend enough money from our entertainment `budget` to make them happy. So what we will do is this: consider every $2^{num\_admitted}$ combinations of customers from the `admitted` array, and admit that combination of customers whose total fame is maximal, but whose total `wait_time` $\leq$ `budget`. In the vocabulary of the knapsack problem, each customer's value is his fame, and his waiting time is his weight. We want the combination of customers with maximum value and with total weight at most `budget`.

The function takes the following arguments, in this order:

- `admitted`: an array of `Customer` structs
- `num_admitted`: the length of the `admitted` array
- `budget`: the maximum total wait time we can accommodate in the restaurant

The return value of the function encodes which customers from `admitted` were allowed to sit. Specifically, bit #`i` is 1 if `admitted[i]` was permitted to sit. Otherwise, the bit is 0. Bits `num_admitted` through 31, inclusive, are always 0. In cases where there are two or more combinations that are maximial, your function may return any of them.

Returns in `$v0`:

- a bitstring (integer) which encodes which customers were seated, or

- `-1` if `num_admitted` ≤ 0 or `budget` ≤ 0

Returns in `$v1`:

- the total fame of the people who are seated; or
- `-1` if `num_admitted` ≤ 0 or `budget` ≤ 0

Additional requirements:

- The function must not write any changes to main memory.

The knapsack problem falls into a class of problems in computer science that cannot be solved efficiently. The number of combinations to consider grows exponentially in the number of admitted customers. This will be taken into account during grading (e.g., when setting maximum instruction counts). We will also assume that `num_admitted` ≤ 30.

Example #1:

```
admitted =
    000: id: 324, fame: 730, wait_time: 19 (priority: 920)
    001: id: 643, fame: 661, wait_time: 4 (priority: 701)
    002: id: 142, fame: 348, wait_time: 28 (priority: 628)
    003: id: 353, fame: 525, wait_time: 9 (priority: 615)
    004: id: 554, fame: 411, wait_time: 20 (priority: 611)
    005: id: 43, fame: 36, wait_time: 17 (priority: 206)
num_admitted = 6
budget = 17
```

Return value in `$v0`: `10`   binary: `00000000000000000000000000001010`

Return value in `$v1`: `1186`

Example #2:

```
admitted =
    000: id: 952, fame: 639, wait_time: 9 (priority: 729)
    001: id: 423, fame: 604, wait_time: 5 (priority: 654)
    002: id: 493, fame: 468, wait_time: 6 (priority: 528)
    003: id: 232, fame: 223, wait_time: 30 (priority: 523)
    004: id: 675, fame: 2, wait_time: 24 (priority: 242)
    005: id: 838, fame: 45, wait_time: 2 (priority: 65)
num_admitted = 6
budget = 25
```

Return value in `$v0`: `39`   binary: `00000000000000000000000000100111`

Return value in `$v1`: `1756`

Example #3:

```
admitted =
    000: id: 298, fame: 794, wait_time: 23 (priority: 1024)
    001: id: 788, fame: 982, wait_time: 3 (priority: 1012)
    002: id: 928, fame: 534, wait_time: 27 (priority: 804)
    003: id: 792, fame: 658, wait_time: 8 (priority: 738)
    004: id: 588, fame: 598, wait_time: 10 (priority: 698)
    005: id: 49, fame: 593, wait_time: 4 (priority: 633)
    006: id: 1000, fame: 306, wait_time: 5 (priority: 356)
    007: id: 830, fame: 83, wait_time: 15 (priority: 233)
    008: id: 20, fame: 192, wait_time: 3 (priority: 222)
    009: id: 649, fame: 169, wait_time: 5 (priority: 219)
num_admitted = 10
budget = 40
```

Return value in `$v0`: `890`   binary: `00000000000000000000001101111010`

Return value in `$v1`: `3498`

Example #4:

```
admitted =
    000: id: 329, fame: 959, wait_time: 22 (priority: 1179)
    001: id: 908, fame: 934, wait_time: 15 (priority: 1084)
    002: id: 954, fame: 782, wait_time: 19 (priority: 972)
    003: id: 815, fame: 857, wait_time: 4 (priority: 897)
    004: id: 588, fame: 696, wait_time: 9 (priority: 786)
    005: id: 530, fame: 705, wait_time: 0 (priority: 705)
    006: id: 190, fame: 562, wait_time: 13 (priority: 692)
    007: id: 227, fame: 306, wait_time: 26 (priority: 566)
    008: id: 812, fame: 352, wait_time: 18 (priority: 532)
    009: id: 231, fame: 387, wait_time: 13 (priority: 517)
    010: id: 543, fame: 187, wait_time: 7 (priority: 257)
    011: id: 632, fame: 92, wait_time: 12 (priority: 212)
num_admitted = 12
budget = 40
```

Return value in `$v0`: `1082`   binary: `00000000000000000000010000111010`

Return value in `$v1`: `3379`

Example #5:

```
admitted =
    000: id: 508, fame: 15, wait_time: 9 (priority: 105)
    001: id: 678, fame: 5, wait_time: 9 (priority: 95)
    002: id: 91, fame: 7, wait_time: 8 (priority: 87)
    003: id: 996, fame: 8, wait_time: 7 (priority: 78)
    004: id: 819, fame: 11, wait_time: 5 (priority: 61)
    005: id: 880, fame: 7, wait_time: 5 (priority: 57)
    006: id: 209, fame: 12, wait_time: 4 (priority: 52)
    007: id: 975, fame: 8, wait_time: 2 (priority: 28)
num_admitted = 8
budget = 20
```

Return value in `$v0`: `209`   binary: `00000000000000000000000011010001`

Return value in `$v1`: `46`

Academic Honesty Policy

Academic honesty is taken very seriously in this course. By submitting your work to Blackboard you indicate your understanding of, and agreement with, the following Academic Honesty Statement:

1. I understand that representing another person's work as my own is academically dishonest.
2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
5. I understand that passing information that is relevant to a homework/exam to others in the course (either lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the College of Engineering and Applied Sciences.
7. I understand that the penalty for academic dishonesty might not be immediately administered. For instance, cheating in a homework may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty.

How to Submit Your Work for Grading

To submit your `proj4.asm` file for grading:

1. Login to Blackboard and locate the course account for CSE 220.
2. Click on "Assignments" in the left-hand menu and click the link for this assignment.
3. Click the "Browse My Computer" button and locate the `proj4.asm` file. Submit only that one `.asm` file.
4. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the steps again. We will grade only your last submission.