Congratulations! Simplicio and Sagredo (you) have landed a summer internship at Facebook’s internal tech division. And, by a stroke of good luck you are both on the same team!

Your project involves dealing with a lot of employee data, so you need a way to be able store employee records, so that you can access them in constant time. Your mentor, Salviati has already created a generic hash table library, which allows a value `elem` to be stored based on it’s unique `key`.

In order to use the hash table, you need to provide the client code. The employee information is represented by the following struct -

```c
struct employee_info{
    int emp_id;
    string first_name;
    string last_name;
    int wage;
};
typedef struct employee_info* employee;
```

It’s obvious that `employees` will be stored as values in the hash table. Given the fields above, what do you think is the appropriate key?

`emp_id`

Now fill in the required information below that the client (you) need to provide the library -

```c
//set elem to employee
typedef employee elem;

//set the key type
typedef int key;
key elem_key(elem e){
    return e->emp_id;
}
bool key_equal(key k1, key k2){
    return k1 == k2;
}
```
Now you need to do some analysis on the pay that employees are receiving. _Salviati_ gives you an array of _n_ employees sorted by wage. The wages are unique. You decide to create a Binary Search Tree of the data, for use in the analysis.

Again, _elem_ is _employee_. But the key might not be the same. What is the appropriate key for the BST?

_wage_

This is what _Simplicio_ has come up with for creating the BST -

```c
1 bst make_tree(employee[] E, int n){
2    bst B = alloc(struct bst_header);
3    for(int i = 0; i < n; i++){
4        bst_insert(B, E[i]);
5    }
6    return B;
7 }
```

What’s a problem with this code?

**Solution:** It results in a worst-case BST that looks like a linked list

Fill in the following code that returns a BST close to the optimum case -
(Hint: Keep in mind the algorithm for binary search and remember that the array is sorted by wage)

```c
1 bst make_tree(employee[] E, int n){
2    bst B = alloc(struct bst_header);
3    B->root = create(E, 0, n);
4    return B;
5 }
6
7 tree* create(employee[] E, int lower, int upper){
8    if(lower == upper) return NULL;
9    int mid = lower + (upper - lower)/2;
10   tree* T = alloc(struct tree_node);
11   T->data = E[mid];
12   T->left = create(E, lower, mid);
13   T->right = create(E, mid + 1, upper);
14   return T;
15 }
```
Now, you’ve been told that your actual assignment involves finding the employee with the lowest wage in the company. Using the BST would involve $O(\log(n))$ time. So, it’s better to use a priority queue that can do it in $O(1)$ time.

Assume that you have created a priority queue $H$ which has wage as the key and employee as the elem.

You are given the state legal minimum wage $\text{min} \_ \text{wage}$. You need to write a function that raises the pay of any employees whose current pay is below the legal minimum wage to the legal minimum wage.

\text{e.g} If the wages of employees A, B, C and D are $(3, 1, 4, 2)$ and the $\text{min} \_ \text{wage}$ is 3, after you run your function, wages should be $(3, 3, 4, 3)$ respectively.

Fill in the the blanks below to create the required function.

1. \begin{verbatim}
   void adjust_wages(heap H, int min_wage){
   while(elem_key(pq_min(H)) < min_wage){
      //remove the element with the minimum wage
      employee temp = pq_delmin(H);
      //replace temp’s wage with the minimum wage
      temp->wage = min_wage;
      //insert temp back into the heap
      pq_insert(H, temp);
   }
   }
\end{verbatim}