Example sheet 7

Amortized analysis. Heaps. Disjoint sets.
Algorithms—DJW[*]—2018/2019

Questions labelled ∘ are warmup questions. Questions labelled ∗ involve more thinking and you are not expected to tackle them all.

Question 1°. Consider a stack that, in addition to `push()` and `pop()`, supports `flush()` which repeatedly pops items until the stack is empty. Explain why the amortized cost of each of these operations is $O(1)$.

Question 2. Consider a k-bit binary counter. This supports a single operation, `inc()`, which adds 1. When it overflows i.e. when the bits are all 1 and `inc()` is called, the bits all reset to 0. The bits are stored in an array, $A[0]$ for the least significant bit, $A[k-1]$ for the most significant.

(i) Give pseudocode for `inc()`.
(ii) Explain why the worst-case cost of `inc()` is $O(k)$.
(iii) Starting with the counter at 0, what is the aggregate cost of m calls to `inc()`? *[Hint. How many times can each bit get flipped? See the footnote on page 45 of lecture notes.]*
(iv) Let $\Phi(A)$ be the number of 1s in A. Use this potential function to calculate the amortized cost of `inc()`.

Question 3. Consider the dynamic array from section 7.2 in lecture notes, but suppose that when the array needs to be expanded we expand the capacity by a factor $k > 1$ instead of doubling. What is wrong with the following argument?

"Define the potential to be $\Phi = 2n - \ell$, where n is the number of items and ℓ is the capacity, with the special case $\Phi = 0$ when $n = 0$. In the case where we don't need to expand the array, true cost is $O(1)$ and $\Delta\Phi = 2$ so amortized cost is $O(1)$. In the case where we do need to expand the array, say from $\ell = n$ to $\ell = kn$, true cost is $O(n)$ and $\Delta\Phi = 2\Delta n - \Delta\ell = 2 - (k-1)n$, so the amortized cost is $O\big(n + (1-k)n\big) = O\big((2-k)n\big)$. If $k \geq 2$ the amortized cost is $O(1)$, and if $k < 2$ the amortized cost is $O(n)$."

Question 4*. Consider a dynamically-sized array that supports appending an element on the right, and deleting the rightmost element. Suppose that the array's capacity is doubled when it becomes full, and halved when it becomes less than 25% full; and that the cost of these resizing operations is $O(n)$ where n is the number of items to copy across into the new array. Using the potential method, show that the append and delete operations both have $O(1)$ amortized cost. What would go wrong if we halved capacity as soon as the array became less than 50% full, rather than 25% full?

Question 5. In a binomial heap with n items, show that the amortized cost of `push()` is $O(1)$. *[Hint: use your answer to Question 2(iv)].*

Question 6°. Consider a binary heap. Find a sequence of N items such that inserting them all takes $\Omega(N \log N)$ elementary operations.

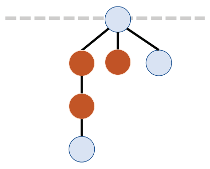
Question 7. An engineer friend tells you excitedly that they have been studying the binary heap and they have found a cunning potential function that proves that the amortized costs of `push` and

[*]Question 9 is from Prof Stajano. Question 14 is from Alstrup and Rauhe, via Inge Li Gortz.

`popmin` are $O(1)$ and $O(\log N)$ respectively. (The big-O expressions are asymptotic in N, the number of items in the heap.)

Given N, construct a sequence comprising m_1 `push` and m_2 `popmin` operations, for which the heap size never exceeds N and for which the total cost is $\Omega\big((m_1+m_2)\log N\big)$. Hence explain carefully why your friend is mistaken. *[Hint. You can choose m_1 and m_2 freely, and they are allowed to depend on N. Use your answer to question 6.]*

Question 8°. Give a sequence of operations that would result in a Fibonacci heap of this shape. (The three darker nodes are losers.) What is the shortest sequence of operations you can find?



Question 9. In the Fibonacci heap, suppose that `decreasekey()` only cuts out (if necessary) the node whose key has been decreased, i.e. it doesn't use the `loser` flag and it doesn't recursively inspect the node's parents. Show that it would be possible for a node with n descendants to have $\Omega(n)$ children.

Question 10°. Prove the result from Section 7.6 of the handout, namely, that in a Fibonacci heap with n items the maximum degree of any node is $O(\log n)$. *[Hint. Use the theorem from that section.]* Must it be a root node that has maximum degree?

Question 11*. Design a priority queue that has bounded size, i.e. given a maximum size M it only keeps the M smallest items. Can you achieve amortized costs for `push` and `decreasekey` of $O(1)$, and for `popmin` of $O(\log M)$?

Question 12. In the flat forest implementation of DisjointSet, using the weighted union heuristic, prove the following:
(i) After an item has had its 'parent' pointer updated k times, it belongs to a set of size $\geq 2^k$.
(ii) If the DisjointSet has n items, each item has had its 'parent' pointer updated $O(\log n)$ times.
(iii) Starting with an empty DisjointSet, any sequence of m operations of which n are `add_singleton()` takes $O(m + n \log n)$ time in aggregate.

Question 13. Sketch out how you might implement the lazy forest DisjointSet, so as to efficiently support "Given an item, print out all the other items in the same set". Explain carefully how `get_set_with` and `merge` are implemented.

Question 14*. Consider an undirected graph with n vertices, where the edges can be coloured blue or white, and which starts with no edges. The graph can be modified using these operations:

- `insert_white_edge(u,v)` inserts a white edge between vertices u and v
- `colour_edges_of(v)` colours blue all the white edges that touch v
- `colour_edge(u,v)` colours the edge $u \leftrightarrow v$ blue
- `is_blue(u,v)` returns True if and only if the edge $u \leftrightarrow v$ is blue

Give an efficient algorithm that supports these operations, and analyse its amortized cost.

Extend your algorithm to also support the following operation, and analyse its amortized cost:

- `are_blue_connected(u,v)` returns True if and only if u and v are connected by a blue path

Extend your algorithm to also support the following operation, and analyse its amortized cost.

- `remove_blue_from_component(v)` deletes all blue edges between pairs of nodes in the blue-connected component containing v

[Note. It's easy to gloss over difficulties, so be sure to be explicit about all operations. If you change your data structure to answer a later part, make sure your earlier answers are still complete. This is the sort of question you might be asked in a Google interview; the interviewer will be looking for you to take ideas that you have been taught and to apply them to novel situations.]