correctness **April 30, 2025**

Homework 4

Spring 2025 *COSC 31: Algorithms*

1. **Merge Sort**

(a) State the pre-condition and the post-condition for the *merge-sort-helper* method so that they imply the correctness of the *merge-sort* method.

Answer

(a) Preconditions:

- $\ell, r \in \mathbb{N}$
- $a[\ell \ldots r]$ is an array of real numbers

(b) Postconditions:terminates, and $a'[\ell \ldots r]$ is a sorted permutation of $a[\ell \ldots r]$.

(b) Write the loop invariant for the while-loop in the *merge* method. The loop invariant you state must be strong enough to imply the correctness of *merge*.

Answer

- $x \in [i, j+1]$
- $y \in [j+1, k+1]$
- Since $z - i = (x - i) + (y - (j+1)) = x + y - j - 1$;
- $b[i \ldots z-1] = b[i \ldots x+y-j-2]$ has a sorted permutation from $a[i \ldots x-1]$ and $a[j+1 \ldots y-1]$
- Elements in $b[i \ldots z-1]$ are at most less or equal to the elements in $a[x \ldots j]$ and $a[y \ldots k]$.
- Array a remains unchanged outside of the range $i \ldots k$.

(c) Prove by induction that your loop invariant is correct.

Answer

Base case: For the first iteration, $x = i$, and $y = j+1$. Then

$$z = x + y - j - 1 = i + j + 1 - j - 1 = i$$

Since no elements have been merged yet, $b[i \ldots i-1]$ is empty array because the ranges $a[i \ldots i-1]$, and $a[j+1 \ldots j]$ are empty.
Inductive case: For the inductive hypothesis, we assume for $x = d$, $y = b$, and $z = c$, the following are true:

- $x \in [d, j+1]$
- $y \in [b, k+1]$

- $z = c$
- $b[i \dots c-1]$ is a sorted permutation from $a[i \dots x-1]$ and $a[j+1 \dots y-1]$
- Elements in $b[i \dots c-1]$ are at most less or equal to the elements in $a[d \dots j]$ and $a[b \dots k]$.
- Array a remains unchanged outside of the range $i \dots k$.

We prove that the loop invariant is true for one more iteration that's either $x = d+1$, $y = b$, $z = c{+}1$ or $x = d$, $y = b{+}1$, $z = c{+}1$ because x and y are incremented alternatingly. Consider the cases:

- $a[d] \leq a[b]$:
  - $x = d+1$, we see that $d+1 \in [d, j+1]$, which is true by inductive hypothesis.
  - $y = b$, we see that $b \in [b, k+1]$ is true by the inductive hypothesis.
  - $b[i \dots c+1-1] = b[i \dots c]$. By the inductive hypothesis, $b[i \dots c-1]$ is sorted permutation of elements in a, and $a[d]$ from subarray $a[d, j]$ is appended to the sorted array $b[i \dots c-1]$. By inductive hypothesis, we see that all elements in $a[d \dots j]$ are greater than or equal to all the elements in $b[i \dots c-1]$ hence $b[i \dots c]$ is sorted.
- $a[d] > a[b]$: we expect $x = d, y = b+1$, and $z = c+1$::
  - $x = d$, we see that $d \in [d, j+1]$ which is true by the inductive hypothesis.
  - $y = b+1$, we see that $b+1 \in [b, k+1]$, which is true by inductive hypothesis.
  - Similarly, $b[i \dots c+1-1] = b[i \dots c]$. By the inductive hypothesis, $b[i \dots c-1]$ is permutation of elements in a, and $a[b]$ from array $a[b, k]$ is appended to the sorted array $b[i \dots c-1]$. By inductive hypothesis, we see that all elements in $a[b \dots k]$ are greater than or equal to all the elements in $b[i \dots c-1]$ hence $b[i \dots c]$ is sorted.

In both cases, we see that a is unchanged outside of the range $i \dots k$, hence, we see that the loop invariant is true for $x = a+1$, $y = b$, and $z = c+1$ or $x = a$, $y = b+1$, and $z = c+1$.

(d) Assuming that the *merge* method is correct, prove by induction that the *merge-sort-helper* method is correct

Answer

Let $P(n)$ be predicate that is true if *merge-sort-helper* meets the precondition, and postconditions in part (a). We need to prove that $\forall n \in \mathbb{N} : P(n)$ is true. We can proceed by strong induction on the size of the subarray $s = r - l + 1$.

<u>base case</u>: We see that if $s = 0$ or $s = 1$, the *merge-sort-helper* terminates without an action. When $s = 2$, we see that $\ell < r$, and $a[\ell]$ and $a[r]$ are sorted by the merge method. Both the precondition, and the postconditions are satisified, and this establishes the base case.

<u>Inductive case</u>: Fix $k \geq 2$, and assume that $P(t)$ is true for $2 \leq t \leq k$, where t corresponds to the size of the subarray s. We need to show that $P(k+1)$ is true. To that end, fix

any array $m[\ell \dots r]$ of size $k+1$, we see that $r > \ell$, and the method executes the recursive steps. Let $mid = \lfloor \frac{\ell+r}{2} \rfloor$. We have the subarrays $m[\ell \dots mid]$, and $m[mid+1 \dots r]$, whose size are exacty equal mid, and $mid \leq k$. By the inductive hypothesis, recursive calls to *merge-sort-helper* on each of these subarrays correctly sort them. Then the merge function is called. By the assumption, *merge* correctly merges two sorted subarrays into one sorted array. Thus, $m'[\ell \dots r]$ is a sorted permutation of $m[\ell \dots r]$. This establishes that $P(k+1)$ is true, and $\forall n \in \mathbb{N} : P(n)$ is true.

(e) Write down the recurrence for merge sort with a (brief) explanation of how you get the recurrence, and solve for the asymptotic time complexity of merge sort.

Answer

The recurrence for the merge sort is $T(n) = 2T(\frac{n}{2}) + O(n)$, because we have two subproblems of size $\frac{n}{2}$ at node of the tree. The combine time is $O(n)$, because in the *merge*, we are comparing the two sorted arrays, who combined size is at most n, and merging them and copying over to the original array to sort in place.

To solve the recurrence, we can use the master theorem.

$$f(n) = O(n)$$
$$n^{\log_2 2} = n$$

We see that this fits into case 2, where $k = 0$, hence we have $T(n) = \Theta(n \log n)$.

2. **Missing Number**

Suppose that $a[1 \dots n]$ contains all but one of the numbers in $\{1, 2, \dots, n$ and $\perp$, in some arbitrary order. Thus, exactly one number from $\{1, 2, \dots, n\}$ is missing in $a[1 \dots n]$. For example, suppose that $n = 5$ and $a[1 \dots 5]$ contains $(4, 5, 1, \perp, 3)$; then, the missing number is 2.

Answer

Algorithm MissingNumber

Precondition: $n \geq 1$; $a[1 \dots n]$ is a permuation of $\{\perp, 1, 2, \dots, n\} - m$, for some $m \in \{1, 2, \dots, n\}$
Postcondition: Terminates, and returns m, and the array a is unchanged outside of the range $1 \dots n$

Function `MissingNumber(`a, n`):`
- expectedSum $\leftarrow \frac{n(n+1)}{2}$
- actualSum $\leftarrow 0$
- **for** $i \leftarrow 1$ **to** n **do**
  - **if** $a[i] \neq \perp$ **then**
    - actualSum $\leftarrow$ actualSum + $a[i]$
  - **end**
- **end**
- **return** expectedSum - actualSum

Correctness: The observation is that, there can only be one missing number from the array $a[1 \dots n]$, hence the difference between the sum of all numbers from $1 \dots n$ and the sum of all numbers in the array a expect for $\perp$ is the missing number. The algorithm runs in $O(2n) = O(n)$

to compute the two sums, and returns the difference.