Workshop 3

In this workshop we will study the MergeSort sorting algorithm. A pseudocode for MERGESORT is presented as figure 1 and can also be looked up in [Ros], section 5.4.4. MERGESORT uses a subroutine MERGE, that is presented in figure 2.

Note that the algorithms are described here a little different than in [Ros], but the underlying idea is the same. To sort a list $L = (a_0, a_1, \ldots, a_{n-1})$ with the program given in the pseudocode one would call the procedure MERGESORT$(L, 0, n-1)$.

Before you start it might be useful to reacquaint yourself with section 5.4.4.

procedure MERGESORT$(L = (a_0, a_1, \ldots, a_{n-1}), l, r)$
  if $l < r$ **then**
    $m = \lfloor \frac{r+l}{2} \rfloor$
    MERGESORT(L, l, m)
    MERGESORT$(L, m+1, r)$
    $L =$ MERGE(L, l, m, r)
  return L

Figure 1: Mergesort

Exercise 1

1. Implement MERGE and MERGESORT.

```
void Merge(int L[], int start, int end, int mid){
  // Allocate space for L_1 and L_2, and populate them with using a for
loop
  int* L_1 = malloc(sizeof(int) * (mid + 1));
  int count_1 = 0;
  for (int i = start; i <= mid; i++) {
    L_1[count_1] = L[i];
    count_1++;
  }
  int* L_2 = malloc(sizeof(int) * (mid));
  int count_2 = 0;
  for (int i = mid + 1; i <= end; i++) {
    L_2[count_2] = L[i];
```

```
    count_2++;
  }



  int i = 0;
  int j = 0;
  while (i < (mid - start + 1) && j < (end - mid)) {
    if (L_1[i] <= L_2[j]) {
      L[start + i + j] = L_1[i];
      i++;
    } else {
      L[start + i + j] = L_2[j];
      j++;
    }
  }
  if (i == mid - start + 1) {
    for (int k = j; k <= end - mid - 1; k++) {
      L[start + i + k] = L_2[k];
    }
  } else {
    for (int k = i; k <= mid - start; k++) {
      L[start + j + k] = L_1[k];
    }
  }
  free(L_1);
  free(L_2);
}

void MergeSort(int L[], int start, int end){
  if (start < end) {
    int mid = (start + end) / 2;
    MergeSort(L, start, mid);
    MergeSort(L, mid + 1, end);
    Merge(L, start, end, mid);
  }
}
```

2. Use MERGESORT to sort the list $L = (5, 3, 8, 1, 6, 10, 7, 2, 4, 9)$

`main.c`:

```
int main(void){
  int L[] = { 5, 3, 8, 1, 6, 10, 7, 2, 4, 9 };
```

```
  int L_size = sizeof(L) / sizeof(L[0]);

  printf("Given list \n");
  printList(L, L_size);

  MergeSort(L, 0, L_size - 1);

  printf("The sorted list \n");
  printList(L, L_size);
  return 0;
}
```

`stdout`:

```
code/DTG/workshop3 on  main [!?] via C v16.0.0-clang
❯ gcc main.c

code/DTG/workshop3 on  main [!?] via C v16.0.0-clang
❯ ./a.out
Given list
5 3 8 1 6 10 7 2 4 9
The sorted list
1 2 3 4 5 6 7 8 9 10
```

Exercise 2

1. Proof by using induction, that MERGESORT returns the sorted list containing the same elements as the input list L. In the proof you may assume, that MERGE returns returns a sorted list containing the elements of two sorted lists given as inputs.

n is the size of the input list L.

Proposition $P(n)$: MergeSort returns the sorted list containing the same elements as the input list L with length n.

Basis step

$P(0)$: here we have an empty list, which is of course trivially sorted.
$P(1)$: this list has 1 element, and is thus trivially sorted of course.

Inductive hypothesis

Assume $P(k)$ for all $k < n$, ie mergesort correctly returns a sorted list containing all input elements.

Inductive step

The case of a list of size n, where $n \geq 2$.

ⓘ MergeSort

MergeSort works by splitting the input list L into two sublists:

- $L_1 = (a_1, a_2, \ldots, a_m)$
- $L_2 = (a_{m+1}, a_{m+2}, \ldots, a_n)$

Where $m = \lfloor \frac{n}{2} \rfloor$. Then MergeSort is called on each half:

- $S_1 = \text{MergeSort}(L_1)$.
- $S_2 = \text{MergeSort}(L_2)$.

Return $\text{Merge}(S_1, S_2)$.

Mergesort splits the input list L into two halves, L_1 and L_2, of sizes $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$ respectively. Since $n \geq 2$, both of these sizes are at least 0 and strictly less than n, so the IH applies. Then, by IH, both mergesorts on L_1 and L_2 return a sorted list containing all elements of L_1 and L_2, respectively. By the assumption that Merge correctly combines two sorted lists, Merge(S_1, S_2) returns a sorted list containing all elements of L, which is what we wanted to show ■

Exercise 3

In [Ros] it is shown that the merging of two sorted lists of lengths a and b can be accomplished by using at most $a + b - 1$ comparisons (Lemma 1, section 5.4.4.).

procedure MERGE(L, l, m, r)
    $L_1 = L[l, l + 1, \ldots, m]$
    $L_2 = L[m + 1, m + 2, \ldots, r]$
    $i = 0$
    $j = 0$
    while $i < m - l + 1$ and $j < r - m$ **do**
        if $L_1[i] \leq L_2[j]$ **then**
            $L[l + i + j] = L_1[i]$
            $i = i + 1$
        else $L[l + i + j] = L_2[j]$
            $j = j + 1$
    if $i = m - l + 1$ **then**
        for $k = j \ldots r - m - 1$ **do**
            $L[l + i + k] = L_2[k]$
    else
        for $k = i \ldots m - l$ **do**
            $L[l + j + k] = L_1[k]$
    return L

1. Assume that MERGE uses (exactly) $a + b - 1$ comparisons to combine two lists with a and b elements. Furthermore, assume that the length of the input list L is $n = 2^k$. Prove by using induction on k, that MERGESORT uses

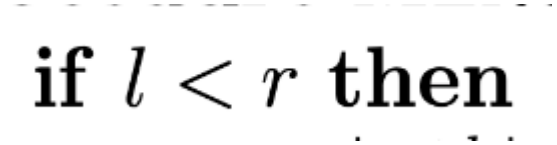
$$n(\log_2(n) + 1) = 2^k(k + 1)$$

comparisons to sort L. What type of induction did you use?

$P(k)$: $\mathrm{MergeSort}$ uses $n(\log_2(n) + 1) = 2^k(k + 1)$ comparisons to sort a list L with length $n = 2^k$.

Basis step

$P(0)$: list L with length $2^0 = 1$
In this case $2^0(0 + 1) = 1(1) = 1$ comparison is used. We can show this in the algorithm definition for $\mathrm{MergeSort}$, as we at this step



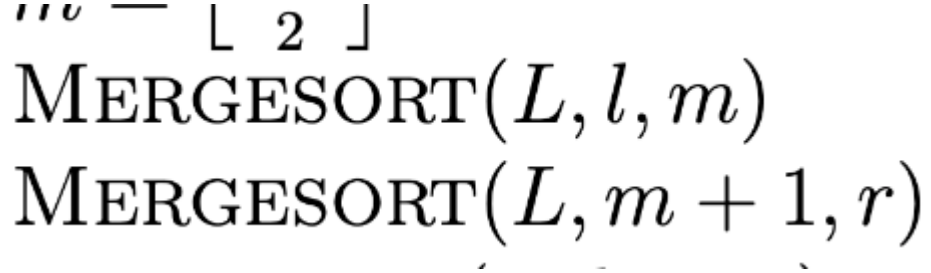
make the comparison $l < r$, which is false, since with a list L of 1 element, the first and last index is the same. Due to this if-condition not going through, we skip directly to the "return L" part of the algorithm, and the algorithm terminates, leaving us with a total of 1 comparison.

Inductive hypothesis

Assume $P(k)$ is true, ie $\mathrm{MergeSort}$ uses $2^k(k + 1)$ comparisons to sort a list L of length 2^k.

Inductive step

Show that $P(k) \rightarrow P(k + 1)$.
In other words, a list L of length 2^{k+1} needs $2^{k+1}(k + 2)$ comparisons to be sorted by merge sort. In the $\mathrm{MergeSort}$ algorithm, we initially use 1 comparison for the initial if-statement, then these calls



each use $2^k(k + 1)$ comparisons per the inductive hypothesis. Then finally the merge step uses per our assumption $a + b - 1$ comparisons, so $2^k + 2^k - 1$ comparisons. Adding these up we get $1 + 2^k(k + 1) + 2^k(k + 1) + 2^k + 2^k - 1$ comparisons, or

$$
\begin{aligned}
&1 + 2^k(k+1) + 2^k(k+1) + 2^k + 2^k - 1 \\
=&1 + 2(2^k(k+1)) + 2(2^k) - 1 \\
=&2(2^k(k+1)) + 2(2^k) \\
=&2^{k+1}(k+1) + 2(2^k) \\
=&2^{k+1}(k+1) + 2^{k+1} \\
=&2^{k+1}(k+2)
\end{aligned}
$$

This fulfills our induction on k, thus proving that $\mathrm{MergeSort}$ uses $2^k(k+1) = n(\log_2(n)+1)$ comparisons to sort a list L with $n = 2^k$ elements. Weak induction (or just induction) was used.

We can prove a similar theorem for lengths n that are not necessarily powers of 2.

2. Use induction to show that MERGESORT uses less or equal to $2n\log_2(n)$ comparisons for all $n \geq 2$ using the same assumptions regarding the MERGE procedure as before.

   Hints: A list with $n+1$ elements is divided into two lists with $\lceil \frac{n+1}{2} \rceil$ and $\lfloor \frac{n+1}{2} \rfloor$ elements respectively by MERGESORT. Furthermore the following in-/equalities might be useful:

   - $2\left\lfloor \frac{n+1}{2} \right\rfloor \log_2(\left\lfloor \frac{n+1}{2} \right\rfloor) \leq 2\left\lfloor \frac{n+1}{2} \right\rfloor \log_2(\left\lceil \frac{n+1}{2} \right\rceil)$

- $\left\lfloor \frac{n+1}{2} \right\rfloor + \left\lceil \frac{n+1}{2} \right\rceil = n+1$
- $\left\lceil \frac{n+1}{2} \right\rceil \leq \frac{2}{3}(n+1)$ for n a positive integer
- $\log_2(\frac{2}{3}(n+1)) = \log_2(n+1) - \log_2(\frac{3}{2})$
- $n + 1 - 2(n+1)\log_2(\frac{3}{2}) < 0$ for positive n.

$P(n)$: $\mathrm{MergeSort}$ uses less or equal to $2n\log_2 n$ comparisons for all $n \geq 2$, assuming Merge uses exactly $a + b - 1$ comparisons, where a and b are the respective lengths of each of the two lists it merges.

Basis step

P(2)

$P(2)$: list L with length 2.
Here $\mathrm{MergeSort}$ uses less or equal to $2n\log_2 n = 2 \cdot 2\log_2 2 = 4$ comparisons. See the above reasoning from Exercise 3.1.

P(3)

$P(3)$: list L with length 3.
Here $\mathrm{MergeSort}$ uses less or equal to $2n \log_2 n = 2 \cdot 3 \log_2 3 \approx 9.51$ comparisons (therefore less or equal to 9). We can show this in the algorithm definition for $\mathrm{MergeSort}$. First the initial if-statement is passed, giving us 1 comparison. Then, the function calls itself on one list of 1 element and one list of 2 elements since the input list of 3 elements is split in two

$$\lfloor 2 \rfloor$$

$$\textsc{MergeSort}(L, l, m)$$

$$\textsc{MergeSort}(L, m+1, r)$$

The merge sort call on the list with 1 element uses 1 comparison, and the call on the list with 2 elements uses 4 comparisons, see above definitions. This brings us to $1 + 1 + 4 = 6$ comparisons. Then when merge is called, $a + b - 1 = 1 + 2 - 1 = 2$ comparisons are used. This brings us to a total of 8 comparisons before the algorithm terminates. 8 comparisons is indeed less or equal to 9 comparisons.

P(4)

$P(4)$: list L with length 4.
Here $\mathrm{MergeSort}$ uses less or equal to $2n \log_2 n = 2 \cdot 4 \log_2 4 = 16$ comparisons. See the above reasoning from Exercise 3.1 – this finds 12 comparisons, which is indeed less than or equal to 16 comparisons. side note: showing this is not needed as a base case, though it does help understand it a little more in my opinion.

Inductive hypothesis

Assume that $P(k)$ is true for all $2 \leq k < n$, in other words assume $\mathrm{MergeSort}$ uses less or equal to $2k \log_2 k$ comparisons to sort a list L with k elements.

Inductive step

Show that $P(k) \rightarrow P(k+1)$
So in the $P(k+1)$ case, we look at whether it's true that a list L of $k+1$ elements takes less or equal to $2(k+1) \log_2 (k+1)$ comparisons to be sorted by $\mathrm{MergeSort}$.

For a list of size $k+1$, $\mathrm{MergeSort}$ divides it into two sublists, which must be of the sizes $\lfloor \frac{k+1}{2} \rfloor$ and $\lceil \frac{k+1}{2} \rceil$.

Let $m = \lfloor \frac{k+1}{2} \rfloor$. The two sublists then have the sizes m and $k + 1 - m$. This follows the fact that $\lfloor \frac{k+1}{2} \rfloor + \lceil \frac{k+1}{2} \rceil = k + 1$.

By the inductive hypothesis, the number of comparisons for the first sublist must at most be $2m \log_2 m$, while the number of comparisons for the second sublist must at most be $2(k + 1 - m) \log_2 (k + 1 - m)$. By our assumption that Merge uses exactly $a + b - 1$ comparisons with a and b as the sizes of the lists being merged, we know that the merge

step must use $m + (k + 1 - m) - 1$ comparisons, which is just k comparisons. With that, the total number of comparisons total out to $2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k$ comparisons. Let's define this total number of comparisons that $\mathrm{MergeSort}$ at most uses to sort a $k + 1$ elements as $T(k + 1)$:

$$T(k + 1) = 2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k,$$

where $m = \lfloor \frac{k+1}{2} \rfloor$. The hints to this exercise give us the following:

$$\lfloor \frac{k + 1}{2} \rfloor + \lceil \frac{k + 1}{2} \rceil = k + 1$$

Which will be useful to simplify $T(k + 1)$. We want to show that

$$T(k + 1) \leq 2(k + 1) \log_2 (k + 1),$$

since showing that will prove $P(n)$: $T(n) \leq 2n \log_2 n$ for all integers $n \geq 2$. Expanded, what we want to show is

$$2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k \leq 2(k + 1) \log_2 (k + 1).$$

Since $m = \lfloor \frac{k+1}{2} \rfloor \leq \frac{k+1}{2}$, can use the fact that $\log_2 (m) \leq \log_2 \frac{k+1}{2}$. Furthermore, $k + 1 - m = \lceil \frac{k+1}{2} \rceil \leq k + 1$. Therefore, $\log_2 (k + 1 - m) \leq \log_2 (k + 1)$. Using these inequalities we can conclude

$$2m \log_2 m \leq 2m \log_2 \frac{k + 1}{2}$$

by multiplying both sides of the first inequality by $2m$, and

$$2(k + 1 - m) \log_2 (k + 1 - m) \leq 2(k + 1 - m) \log_2 (k + 1)$$

by multiplying both sides of the second inequality by $2(k + 1 - m)$. We add both the inequalities together:

$$\begin{aligned} &2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) \\ \leq &2m \log_2 \frac{k + 1}{2} + 2(k + 1 - m) \log_2 (k + 1) \end{aligned}$$

And then add k to both sides:

$$\begin{aligned} &2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k \\ \leq &2m \log_2 \frac{k + 1}{2} + 2(k + 1 - m) \log_2 (k + 1) + k \end{aligned}$$

The left hand side of the inequality is now exactly $T(k + 1)$:

$$\begin{aligned} &2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k \\ = T(k + 1) = &2m \log_2 m + 2(k + 1 - m) \log_2 (k + 1 - m) + k, \end{aligned}$$

So we can simply replace that left hand side:

$$T(k + 1) \leq 2m \log_2 \frac{k + 1}{2} + 2(k + 1 - m) \log_2 (k + 1) + k$$

Now we simplify the right side with the log rule $\log \frac{a}{b} = \log(a) - \log(b)$ and $\log_b(b) = 1$:

$$T(k+1) \leq 2m(\log_2(k+1) - 1) + 2(k+1-m)\log_2(k+1) + k$$

Then distribute that $2m$:

$$T(k+1) \leq 2m\log_2(k+1) - 2m + 2(k+1-m)\log_2(k+1) + k$$

We can then rearrange this

$$T(k+1) \leq 2m\log_2(k+1) + 2(k+1-m)\log_2(k+1) - 2m + k$$

such that it is obvious that $\log_2(k+1)$ may be factored out:

$$T(k+1) \leq \log_2(k+1)(2m + 2(k+1-m)) - 2m + k$$

Distributing the $2(k+1-m)$:

$$T(k+1) \leq \log_2(k+1)(2m + 2k + 2 - 2m) - 2m + k$$

makes it obvious that the $2m$ terms cancel out:

$$T(k+1) \leq \log_2(k+1)(2k+2) - 2m + k$$

after which we can re-pull out the 2:

$$T(k+1) \leq 2(k+1)\log_2(k+1) - 2m + k$$

Finally, to finish the proof, we must show that $-2m + k \leq 0$, ie $k \leq 2m$, ie $k \leq 2\lfloor\frac{k+1}{2}\rfloor$. Here we do a split case. If k is odd, then $k+1$ is even, so $m = \frac{k+1}{2}$, therefore $2m = k+1 > k$. If k is even, then $k+1$ is odd, so $m = \frac{k}{2}$, and $2m = k$. In conclusion, this shows that the term $-2m + k \leq 0$, which means (due to the direction of our inequality) that the term can be dropped. We now have

$$T(k+1) \leq 2(k+1)\log_2(k+1)$$

which is what we set out to prove ■

3. What type of induction did you use? What does the result tell us about the complexity of MERGESORT in big-O notation?

Strong induction was used due to the IH having the assumption apply to $2 \leq k < n$.

In big O notation we drop coefficients and as such the result

$$T(n) \leq 2n\log_2 n$$

tells us that the time complexity of mergesort is within $O(n\log n)$ with respect to comparisons.

$$T(n) \in O(n\log n).$$