

# 1 Workshop 3 - Induktion

I denne workshop kigger vi på MERGESORT, som er en sorteringsalgoritme. Algoritmen i pseudokode kan ses i Figur 1 eller i afsnit 5.4.4 i [Ros]. MERGESORT benytter MERGE, som kan ses i Figur 2 eller i afsnit 5.4.4 i [Ros]. Bemærk at den her er beskrevet på en lidt anden måde, men at metoden og tanken bag er den samme. Inden I begynder på selve opgaverne kan I med fordel læse og forstå afsnit 5.4.4. Som algoritmen er beskrevet herunder vil man starte med at kalde MERGESORT( $L = (a_0, a_1, \dots, a_{n-1})$ , 0,  $n - 1$ ).

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

Figure 1: Mergesort

```
procedure MERGE( $L, l, m, r$ )  
   $L_1 = L[l, l + 1, \dots, m]$   
   $L_2 = L[m + 1, m + 2, \dots, r]$   
   $i = 0$   
   $j = 0$   
  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  $k = j \dots r - m - 1$  do  
       $L[l + i + k] = L_2[k]$   
  else  
    for  $k = i \dots m - l$  do  
       $L[l + j + k] = L_1[k]$   
  return  $L$ 
```

Figure 2: Merge

## 1.1 Delopgave 1

### 1.1.1

**Task:** Implementer MERGE og MERGESORT.

**Solution:** Taget fra <https://www.geeksforgeeks.org/c-program-for-merge-sort/> fordi den fra moodle på ingen måde gad at virke uanset hvor meget jeg rettede i det. Den er alligevel baseret på næsten en direkt kopi af denne, så det går nok.

---

```
#include<stdlib.h>
#include<stdio.h>

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
    }
}
```

```

        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)

```

```

        printf("%d ", A[i]);
    printf("\n");
}

int main()
{
    int arr[] = { 5, 3, 8, 1, 6, 10, 7, 2, 4, 9 };
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}

```

---

### 1.1.2

**Task:** Brug MERGESORT til at sortere listen  $L = (5, 3, 8, 1, 6, 10, 7, 2, 4, 9)$

**Solution:**

---

```

benjamin@DESKTOP-CNN41EU:~/c_opgaver$ ./mergekopi
Given array is
5 3 8 1 6 10 7 2 4 9
Sorted array is
1 2 3 4 5 6 7 8 9 10

```

---

## 1.2 Delopgave 2

**Task:** Bevis ved hjælp af induktion, at MERGESORT returnerer den sorterede liste hvis den får en liste med heltal som input. I beviset antages det, at MERGE returnerer en sammenflettet sorteret liste, hvis dellisterne  $L[l, l + 1, \dots, m]$  og  $L[m + 1, m + 2, \dots, r]$  er sorteret.

**Solution:** In the base case we have an array with 1 element when we call mergesort we get the following:

---

```
mergeSort(arr, 0, arr_size - 1);
```

---

With actual values we get the following.

---

```
mergeSort(arr, 0, 0);
```

---

in the function we get that  $l = 0$  and that  $r = 0$  the first thing we check is that  $l < r$  Since this is not true an the function doesn't continue and thus the array is sorted which is true. In the case where we have an array size of 2 we split the elements into two arrays with ane element in each, and call mergeSort an each of them. Since both only have one element we return to the original function, and call merge. Merge then gives us the sorted array back.

Induction step: We assume that the mergeSort will give us a single sorted list if we give it the following sorted lists:  $L[l, l + 1, \dots, m]$  and  $L[m + 1, m + 2, \dots, r]$  are sorted. Now if we have give at an unsorted list with  $r+1$  elements it's going to split it in the middle. And we are going to split those and were going to continue to split until we have to lists with one element in each. We can now start to merge, this will go well until we have  $L[l, l + 1, \dots, m]$  and  $L[m + 1, m + 2, \dots, r, r + 1]$  that are sorted. In merge we go through we have iterators for each list. Then it compares the element indexed at each element. if the element in the first list is smaller than the second then the first iterator goes up and vice versa.

---

```
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
```

```

        j++;
    }
    k++;
}

```

---

If last element of the first list is smaller than a  $r$  or an element before that we have just append the last list to the list we want. If it turns out that the value at index  $r+1$  is smaller than some value in the first list we will have that  $r+1$  will be the last element put in the bigger list before we append the last values from the first list.

---

```

/* Copy the remaining elements of L[], if there
are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there
are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

```

---

And thus the original list with  $r+1$  elements is sorted.

### 1.3 Delopgave 3

Lemma 1 i afsnit 5.4.4 i [Ros] siger, at to sorterede lister med  $m$  og  $n$  elementer kan sammenflettes (merged) til en sorteret liste ved højst  $m + n - 1$  sammenligninger.

#### 1.3.1

**Task:** Antag at MERGE benytter  $m + n - 1$  sammenligninger til at sammenflette to lister med  $m$  og  $n$  elementer. Antag yderligere, at  $n = 2^k$  og vis ved induktion over  $k$  (start med  $k = 0$ ), at MERGESORT bruger præcis

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

sammenligninger hvis input-listen  $L$  har  $n = 2^k$  elementer. Hvilket slags induktionsbevis brugte du?

**Solution:** If we start in the base case where  $k = 0$  we have that  $n = 1$ . When we only have one element MERGESORT only makes one comparison to see if  $l < r$ . Now we have to show the following:

$$\begin{aligned} 2^0(0 + 1) &= 1 \\ 1 \cdot 0 + 1 \cdot 1 &= 1 \end{aligned} \tag{1}$$

Now for the inductive step where where our list has  $k + 1$  elements. When  $k$  goes up by one we double our number of elements. This means that when we split our list the first time we get to lists with  $k$  elements in each because  $2^{k+1} = 2 \cdot 2^k = 2^k + 2^k$ , Merge sort will be called on both of these which we assume will make  $2^k(k + 1)$  comparisons plus we also call merge which will make  $m + n - 1$  comparisons. we also know that for  $k > 0$   $n$  is an even number meaning that  $m = n$ . Lastly when we call mergesort on the original list we make one comparison to see if  $l < r$ . We can now make the following expression.

$$\begin{aligned} 2 \cdot 2^k(k + 1) + 2n - 1 + 1 \\ 2 \cdot 2^k(k + 1) + 2n \\ 2 \cdot 2^k(k + 1) + 2 \cdot 2^k \\ 2^{k+1}(k + 1) + 2^{k+1} \\ 2^{k+1}((k + 1) + 1) \\ 2^{k+1}(k + 2) \end{aligned} \tag{2}$$

■

We used weak induction on  $k$ , but since  $k$  is doubling  $n$  we might have used string induction on  $n$ . This needs to be researched a little more.

### 1.3.2

**Task:** Lav nu et induktionsbevis for, at MERGESORT bruger mindre end eller lig med  $2n \log_2(n)$  sammenligninger for alle  $n \geq 2$  under samme antagelser om MERGE som ovenfor.

Hints til opgaven: Vi bliver nødt til at lave induktion over  $n$ . Bemærk at en liste med  $n+1$  elementer deles op i to lister med  $\lceil \frac{n+1}{2} \rceil$  og  $\lfloor \frac{n+1}{2} \rfloor$  elementer i Mergesort. Desuden kan I få brug for følgende vurderinger:

- $2 \lfloor \frac{n+1}{2} \rfloor \log_2(\lfloor \frac{n+1}{2} \rfloor) \leq 2 \lfloor \frac{n+1}{2} \rfloor \log_2(\lceil \frac{n+1}{2} \rceil)$
- $\lfloor \frac{n+1}{2} \rfloor + \lceil \frac{n+1}{2} \rceil = \frac{n+1}{2}$
- $\lceil \frac{n+1}{2} \rceil \leq \frac{2}{3}(n+1)$  når  $n$  er et positivt heltal
- $\log_2(\frac{2}{3}(n+1)) = \log_2(n+1) - \log_2(\frac{3}{2})$
- $n+1 - 2(n+1) \log_2(\frac{3}{2}) < 0$  når  $n$  er positiv.

**Solution:** Base step we have that  $n = 2$  and we have to show that the following is true:

$$\begin{aligned} 2(\log_2(2) + 1) &\leq 2 \cdot 2(\log_2(2)) \\ 2(1 + 1) &\leq 2 \cdot (2) \\ 4 &\leq 4 \end{aligned} \tag{3}$$

Now for the inductive step we have to show the following:

$$\begin{aligned} (n+1)(\log_2(n+1) + 1) &\leq 2(n+1)\log_2(n+1) \\ (n+1)\log_2(n+1) + n+1 &\leq 2(n+1)\log_2(n+1) \\ n+1 &\leq (n+1)\log_2(n+1) \\ \frac{n+1}{n+1} &\leq \frac{(n+1)\log_2(n+1)}{n+1} \\ 1 &\leq \log_2(n+1) \end{aligned} \tag{4}$$

■

The last part is true because  $n \geq 2$



### 1.3.3

**Task:** Hvilket slags induktionsbevis brugte du? Og hvad siger resultatet om tidskompleksiteten/store- $O$  for mergesort?

**Solution:** Weak induction was used. It means that the time complexity is  $O(2n\log_2(n))$ . We also have that if  $f_1$  is  $O(x_1)$  and  $f_2$  is  $O(x_2)$  then  $f_1 \cdot f_2$  is  $O(x_1 \cdot x_2)$ .

We have the function  $2n$  and  $\log_2(n)$  the latter can't be shortened but the first can.

$$\begin{aligned} |2n| &\leq C|n| \\ |2n| &\leq 3|n| \\ (3, 0) \end{aligned} \tag{5}$$

Therefore we get a time complexity of  $O(n\log_2(n))$ .