

# Dziel i zwyciężaj

## Sprawozdanie z laboratorium 6 – Piotr Sarna LK1

### Cel ćwiczenia

Podczas zajęć poznaliśmy funkcje działające w sposób rekurencyjny. Wykorzystaliśmy je początkowo do wyliczenia sumy wyrazów ciągu Fibonacciego. Następnie mieliśmy wykorzystać rekurencję w algorytmie sortującym, o nazwie „Merge sort” oraz go zaimplementować.

### Wstęp teoretyczny

Rekurencja występuje, kiedy funkcja wywołuje samą siebie w pętli, aż do osiągnięcia pewnego warunku kończącego, przerywającego cykl wywołań.

Rekurencje możemy podzielić na:

- Bezpośrednią
- Pośrednią
- Liniową
- Nieliniową

Bezpośrednia rekurencja wywołuje sama siebie, bez funkcji pośredniej

Pośrednia rekurencja wywołuje np. funkcję „F2” przez funkcję „F1”, oraz „F1” przez funkcję „F2”

Rekurencja liniowa ma tylko jedno wywołanie samej siebie wewnątrz funkcji.

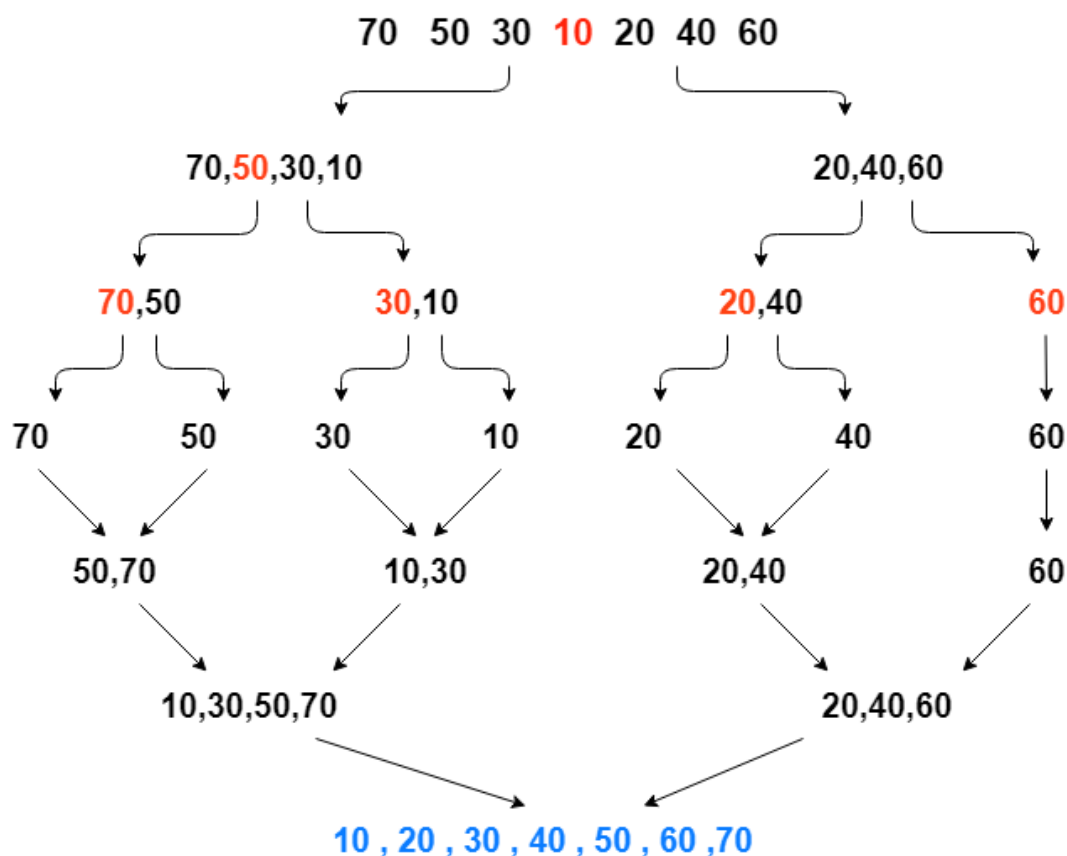
Rekurencja nieliniowa ma większą liczbę wywołań samej siebie wewnątrz funkcji, np. ciąg Fibonacciego.

Zaletą rekurencji jest krótszy i łatwiejszy do zrozumienia kod, jednak zajmuje ona zdecydowanie więcej czasu i miejsca w pamięci w porównaniu z metodą iteracyjną, nawet w przypadku prostych algorytmów.

Rekurencję wykorzystuje się w połączeniu z metodą „dziel i zwyciężaj”, mającej na celu podzielenie problemu na mniejsze części, rozwiązanie go dla jak najmniejszej części a następnie złączenia otrzymanych wyników w celu otrzymania ostatecznego rozwiązania. Przykładem wykorzystania metody „dziel i zwyciężaj” jest algorytm sortujący „Merge sort”, który dzieli tablicę na pół do momentu uzyskania elementu, którego już nie będzie się dało bardziej podzielić a następnie złączenie tych połówek w odpowiedni sposób, dzięki czemu na wyjściu otrzymamy posortowaną tablicę wejściową.

## Opis algorytmu

Sortowanie przez scalanie (ang. Merge sort) polega na rekurencyjnym podziale tablicy wejściowej na pół, do momentu w którym nie będziemy już w stanie bardziej jej podzielić (każda połówka będzie pojedynczym elementem), a następnie złączeniem każdej z takich połówek w odpowiedniej kolejności elementów



Źródło:

<https://www.digitalocean.com/community/tutorials/merge-sort-algorithm-java-c-python>

Merge sort jest rekurencją nieliniową, co ukazuje powyższy schemat.

```
void mergeSort(int* array, int left, int right) {  
    if (left < right) {  
        int mid = (left + right) / 2;  
  
        mergeSort(array, left, mid);  
        mergeSort(array, mid + 1, right);  
        merge(array, left, mid, right);  
    }  
}
```

W każdym wywołaniu funkcji, jeżeli warunek kończący nie jest spełniony (indeks „left” jest mniejszy od „right”, co oznacza, że między nimi znajdują się ciągłe elementy do podzielenia)

wyznaczamy połowę zakresu na którym aktualnie pracujemy (indeks „mid” między „left” i „right”), a następnie wywołujemy rekurencyjnie funkcję dla obu połówek tablicy, dzieląc ją na jeszcze mniejsze połówki. Po podzieleniu tablicy na połówki, złączmy je w następujący sposób:

```
void merge(int* array, int left, int mid, int right) {
    int leftSize = mid - left + 1;
    int rightSize = right - mid;

    int* leftArray = new int[leftSize];
    int* rightArray = new int[rightSize];

    for (int i = 0; i < leftSize; i++) {
        leftArray[i] = array[left + i];
    }
    for (int i = 0; i < rightSize; i++) {
        rightArray[i] = array[mid + 1 + i];
    }

    int leftMergeIndex = 0;
    int rightMergeIndex = 0;
    int mergedIndex = left;

    while (leftMergeIndex < leftSize && rightMergeIndex < rightSize) {
        if (leftArray[leftMergeIndex] < rightArray[rightMergeIndex]) {
            array[mergedIndex] = leftArray[leftMergeIndex];
            leftMergeIndex++;
        }
        else {
            array[mergedIndex] = rightArray[rightMergeIndex];
            rightMergeIndex++;
        }

        mergedIndex++;
    }

    while (leftMergeIndex < leftSize) {
        array[mergedIndex] = leftArray[leftMergeIndex];
        leftMergeIndex++;
        mergedIndex++;
    }

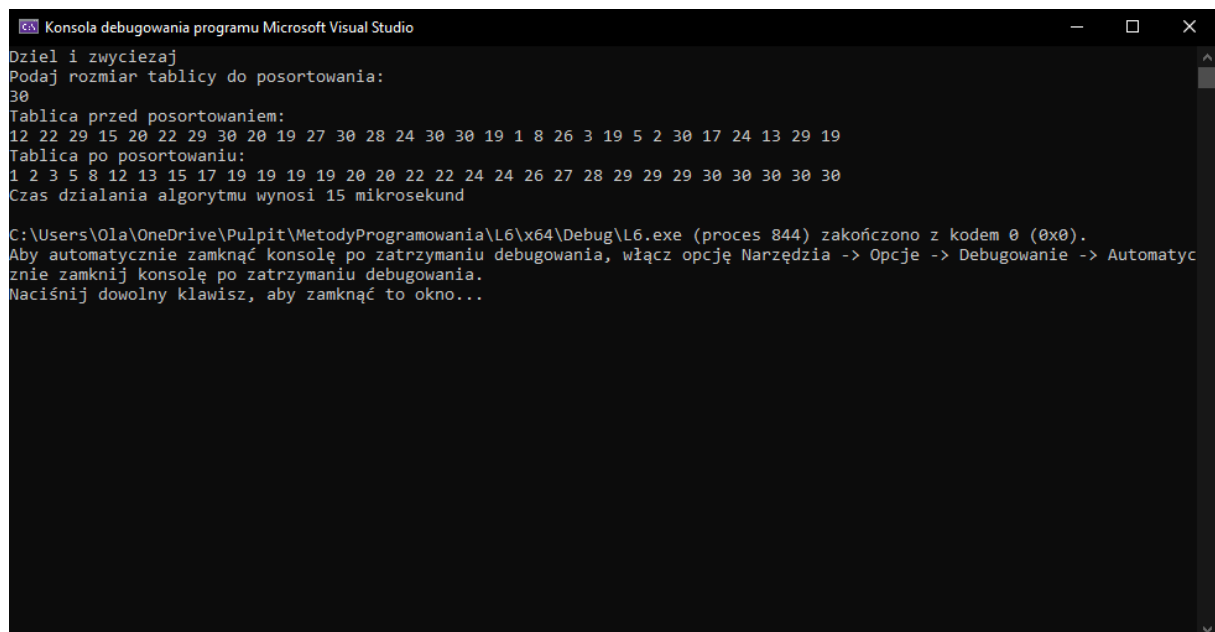
    while (rightMergeIndex < rightSize) {
        array[mergedIndex] = rightArray[rightMergeIndex];
        rightMergeIndex++;
        mergedIndex++;
    }

    delete[] leftArray;
    delete[] rightArray;
}
```

Tworzymy dwie tablice („leftArray” oraz „rightArray”), które będą przechowywać zawartość lewej i prawej połówki po podzieleniu. Następnie do pierwotnej tablicy wstawiamy elementy z obu tablic, porównując je po kolei ze sobą. Jeżeli element z lewej tablicy jest większy od elementu z prawej, wrzucamy go do pierwotnej tablicy a indeks „leftMergeIndex” przesuwamy o 1, ponieważ poprzedni element już umieściliśmy w pierwotnej tablicy. Analogicznie działamy w drugą stronę, jeśli okaże się, że to element tablicy prawej jest większy.

Pod zakończeniu porównywania wrzucamy całą zawartość lewej/prawej tablicy do tablicy pierwotnej, na wypadek, gdyby pozostały w którejś z nich jeszcze jakieś elementy.

Prezentacja działania mojej implementacji w C++:



```
Konsola debugowania programu Microsoft Visual Studio
Dziel i zwyciężaj
Podaj rozmiar tablicy do posortowania:
30
Tablica przed posortowaniem:
12 22 29 15 20 22 29 30 20 19 27 30 28 24 30 30 19 1 8 26 3 19 5 2 30 17 24 13 29 19
Tablica po posortowaniu:
1 2 3 5 8 12 13 15 17 19 19 19 20 20 22 22 24 24 26 27 28 29 29 29 30 30 30 30 30
Czas działania algorytmu wynosi 15 mikrosekund

C:\Users\Ola\OneDrive\Pulpit\MetodyProgramowania\L6\x64\Debug\L6.exe (proces 844) zakończono z kodem 0 (0x0).
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

## Wnioski

Algorytm „Merge sort” mimo, wykorzystania rekurencji jest jednym z najbardziej wydajnych algorytmów sortujących dzięki wykorzystaniu metody „dziel i zwyciężaj”. Cechuje go złożoność  $O(n \log n)$ . Złożoność ta wynika z tego, że tablica dzielona jest na dwie mniejsze części, więc liczba etapów dzielenia jest proporcjonalna do logarytmu z liczby elementów, czyli  $O(\log n)$ . Następnie mniejsze części są ze sobą scalane. Aby je scalić należy przejść przez wszystkie elementy po kolei, co daje złożoność  $O(n)$ . Iloczyn tych dwóch daje złożoność  $O(n \log n)$ .

## Bibliografia

<https://www.korepetycjezinformatyki.pl/rekurencja-czym-jest-w-informatyce/>

<https://www.digitalocean.com/community/tutorials/merge-sort-algorithm-java-c-python>