

# Sortowanie przez kopcowanie

Sprawozdanie z laboratorium 5 – Piotr Sarna LK1

## Cel ćwiczenia

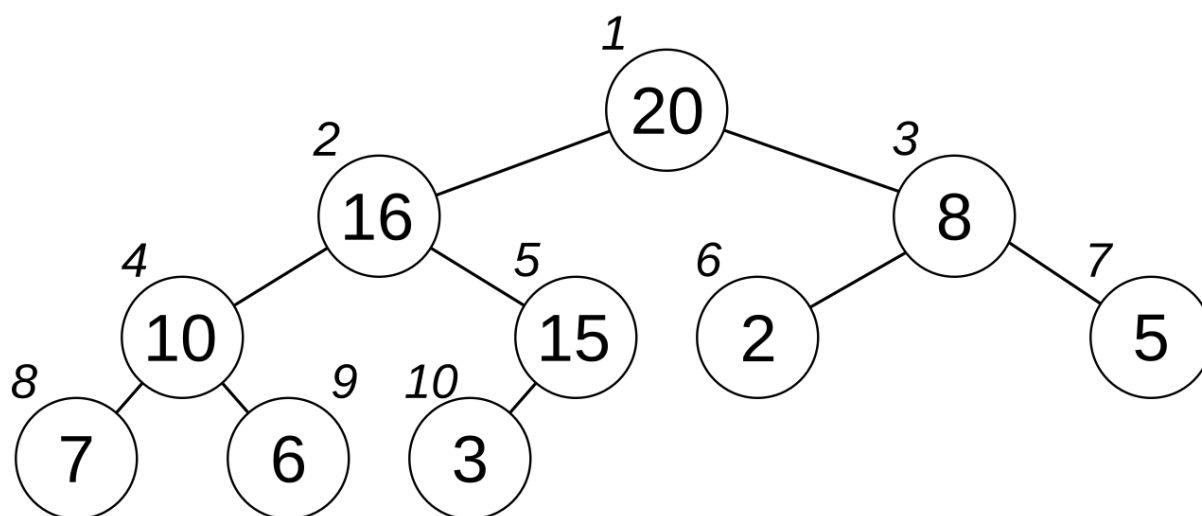
Podczas zajęć zapoznaliśmy się z algorytmami sortującym wykorzystującymi kopiec binarny (ang. Heap Sort). Są to algorytmy, mające na celu posortowanie tablicy, w naszym przypadku na dwa sposoby: niemalejący i nierosnący. Po ich zaimplementowaniu mieliśmy zbadać złożoność obliczeniową.

## Wstęp teoretyczny

Kopiec binarny to specjalny rodzaj drzewa binarnego, które cechuje się następującymi właściwościami:

- Wszystkie poziomy oprócz ostatniego muszą być pełne
- Jeśli ostatni poziom NIE jest pełny, elementy w nim (dzieci) są ułożone od lewej do prawej
- Elementy kopca nazywamy rodzicami oraz dziećmi. Rodzice to elementy znajdujące się nad dziećmi i analogicznie w drugą stronę
- Kopce dzielą się na dwa typy, MAX oraz MIN. W tym pierwszym każde dziecko musi mieć wartość mniejszą lub równą od wartości rodzica, a w drugim przypadku każdy rodzic musi mieć wartość mniejszą lub równą od swojego dziecka

Reprezentacja graficzna kopca binarnego:



Źródło: [https://pl.wikipedia.org/wiki/Kopiec\\_binarny](https://pl.wikipedia.org/wiki/Kopiec_binarny)

Najpopularniejszą formą reprezentacji kopca binarnego jest graf, który przejrzysto pokazuje relacje pomiędzy poszczególnymi elementami. Powyżej znajduje się przykładowy kopiec binarny typu MAX. Liczby w kółkach reprezentują wartości, a liczby obok nich indeksy elementów.

W programie kopiec binarny możemy zaprezentować w postaci tablicy jednowymiarowej:

Indeks	0	1	2	3	4	5	6	7	8	9
Wartość	20	16	8	10	15	2	5	7	6	3

Na zdjęciu indeksowanie zaczyna się od „1”, w mojej implementacji wybrałem jednak indeksowanie od „0”, ponieważ dzięki temu łatwiej mi było obsługiwać kopiec w tablicy w języku C++.

W przypadku takiej implementacji w tablicy łatwo zauważyć pewną zależność. Aby odnaleźć indeks rodzica na podstawie indeksu dziecka, wystarczy wykonać następujące obliczenie:

$$\text{Indeks rodzica} = (\text{Indeks dziecka} - 1) // 2$$

(Zapisałem dzielenie całkowite za pomocą operatora „//”).

Do operacji na kopcu w mojej implementacji wykorzystałem następujące funkcje:

- getParent
- shiftUp
- generateHeap

Funkcja getParent zwraca indeks rodzica na podstawie indeksu dziecka, wykonując obliczenie takie jak zapisane powyżej.

Funkcja shiftUp przywraca własność kopca dla odpowiedniego indeksu dziecka. Porównuje jego wartość z wartością rodzica i w przypadku, gdy wartość dziecka okaże się większa, zamienia je miejscami. Następnie dawny rodzic staje się dzieckiem, a funkcja wykonuje się na nowo do czasu, aż zostanie spełniony warunek „wyższości” rodzica lub dojdziemy do szczytu kopca.

Funkcja generateHeap tworzy kopiec z wygenerowanej losowo tablicy, wykorzystując funkcję shiftUp zapisaną powyżej dla każdego elementu w tablicy.

# Opis algorytmów

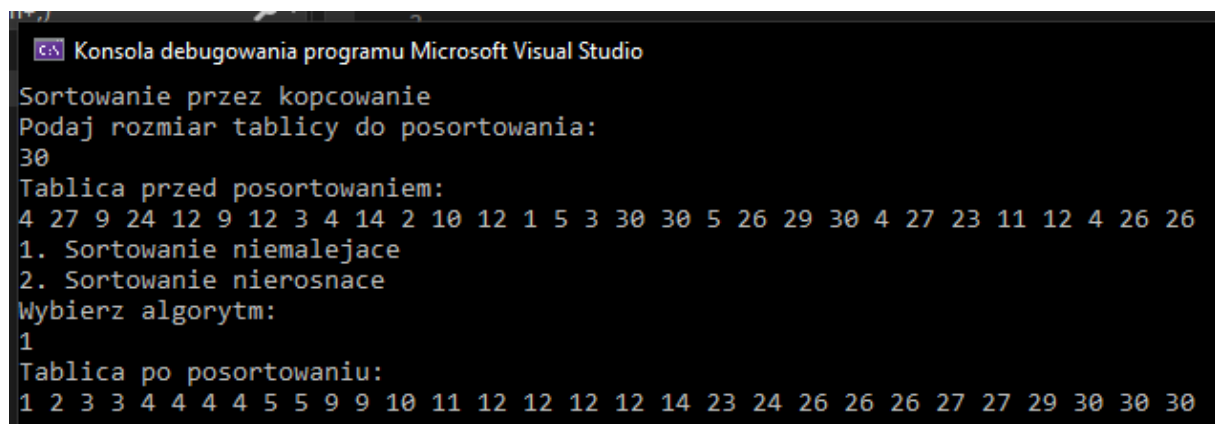
## 1. Algorytm sortujący niemalejąco

Algorytm ten wykorzystuje własność kopca MAX, w którym największy element kopca znajduje się pod indeksem „0”. Zamieniamy go z ostatnim elementem kopca, dzięki czemu mamy odosobniony interesujący nas największy element, zmniejszamy wielkość kopca o 1, aby nie brać już pod uwagę dawnego największego elementu i przywracamy własność kopca w pomniejszonym o 1 zakresie. Powtarzamy to tak długo, aż rozmiar kopca będzie wynosił 1.

Zapis algorytmu w pseudokodzie:

```
utwórz kopiec w A (metodą top-down)
dla i = n-1 do 1 powtarzaj:
    zamień  $A_{n-1}$  z  $A_0$ 
    zmniejsz rozmiar kopca o 1
    przywróć własność kopca metodą top-down
```

Prezentacja działania mojej implementacji w języku C++:

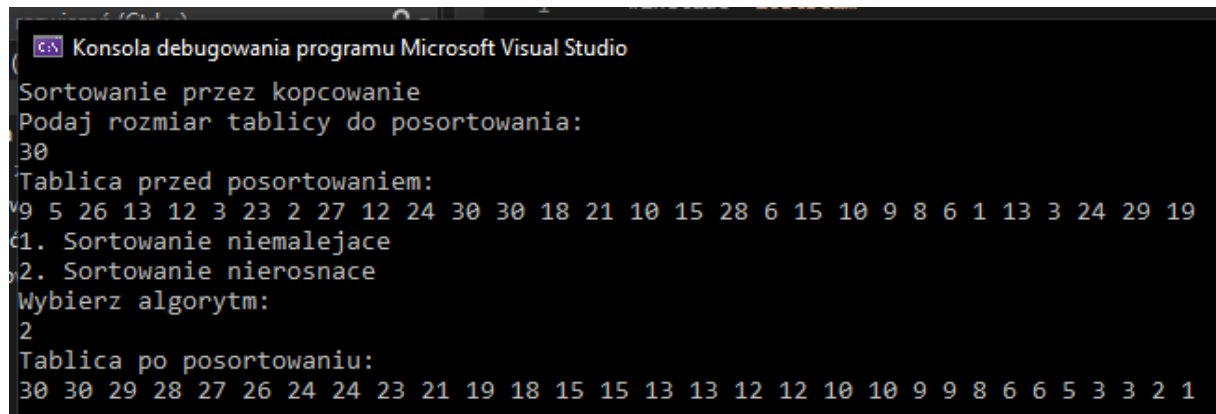


```
Konsola debugowania programu Microsoft Visual Studio
Sortowanie przez kopcowanie
Podaj rozmiar tablicy do posortowania:
30
Tablica przed posortowaniem:
4 27 9 24 12 9 12 3 4 14 2 10 12 1 5 3 30 30 5 26 29 30 4 27 23 11 12 4 26 26
1. Sortowanie niemalejące
2. Sortowanie nierosnące
Wybierz algorytm:
1
Tablica po posortowaniu:
1 2 3 3 4 4 4 4 5 5 9 9 10 11 12 12 12 12 14 23 24 26 26 26 27 27 29 30 30 30
```

## 2. Algorytm sortujący nierosnąco

Aby posortować tablicę nierosnąco, należy wykorzystać kopiec MIN zamiast kopca MAX, reszta kroków pozostaje taka sama.

Prezentacja działania mojej implementacji w języku C++:



```
Konsola debugowania programu Microsoft Visual Studio
Sortowanie przez kopcowanie
Podaj rozmiar tablicy do posortowania:
30
Tablica przed posortowaniem:
9 5 26 13 12 3 23 2 27 12 24 30 30 18 21 10 15 28 6 15 10 9 8 6 1 13 3 24 29 19
1. Sortowanie niemalejace
2. Sortowanie nierosnace
Wybierz algorytm:
2
Tablica po posortowaniu:
30 30 29 28 27 26 24 24 23 21 19 18 15 15 13 13 12 12 10 10 9 9 8 6 6 5 3 3 2 1
```

## Wnioski

Własność kopca zarówno MIN i MAX można łatwo wykorzystać w algorytmach sortujących. Algorytmy te mają dobrą złożoność czasową  $O(n \log n)$ , oraz pamięciową  $O(n)$ . Są one zatem nieco wolniejsze od sortowania szybkiego, lecz mają lepszą pesymistyczną złożoność czasową. Złożoność  $O(n \log n)$  wynika z tego, że w każdej iteracji zmniejszamy rozmiar kopca o 1, ze złożonością  $O(n)$ . Po usunięciu musimy przywrócić własność kopca metodą top-down, która działa w czasie  $O(\log n)$ . Iloczyn tych dwóch działań daje złożoność  $O(n \log n)$ .

## Bibliografia

[https://pl.wikipedia.org/wiki/Kopiec\\_binarny](https://pl.wikipedia.org/wiki/Kopiec_binarny)

[https://eduinf.waw.pl/inf/alg/001\\_search/0113.php](https://eduinf.waw.pl/inf/alg/001_search/0113.php)