

# Algorytmy i Struktury Danych Projekt

Temat: Sortowanie bąbelkowe, quicksort i heapsort

Damian Banasik

Patryk Banaś

## Sortowanie bąbelkowe

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Największą wadą sortowania bąbelkowego jest czas wykonania, który jest szczególnie duży przy pracy z bardzo dużą ilością elementów.

```
void bubblesort(int tab[], int n)
{
    for(int i=1; i<n; i++){
        for(int j=n-1; j>=1; j--){
            if(tab[j]<tab[j-1]){
                int swap;
                swap=tab[j-1];
                tab[j-1]=tab[j];
                tab[j]=swap;
            }
        }
    }
}
```

Przykładowa implementacja sortowania(rosnącego) bąbelkowego.

W programie odbywa się sortowanie tablicy n elementów w której znajdują się wartości typu int.

W celu uzupełnienia tablicy używana jest poniższa funkcja, która losuje liczby z zakresu od 0 do 10 i uzupełnia nimi tablice o wielkości podawanej przez użytkownika.

```
void fill(int tab[], int n)
{
    srand(time(NULL));
    for(int i=0; i<n; i++){
        tab[i]=rand()%11;
    }
}
```

Kolejna funkcja służy do wyświetlania zawartości uzupełnionej wcześniej tablicy.

```
void print_tab(int tab[],int n)
{
    for(int i=0;i<n;i++){
        printf("%d ",tab[i]);
    }
}
```

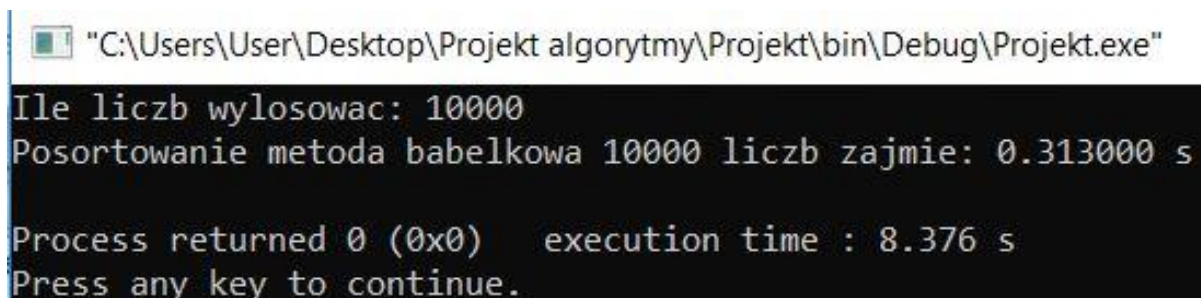
Fragment kodu odpowiedzialny za pobranie od użytkownika ilości liczb które mają zostać wylosowane i umieszczone w tablicy. Następnie wyświetla ile czasu minęło od rozpoczęcia funkcji odpowiedzialnej za sortowanie do jej zakończenia.

```
int n;
printf("Ile liczb wylosowac: ");
scanf("%d", &n);

int tab[n];
fill(tab,n);

start = clock();
bubblesort(tab,n);
stop = clock();
czas = (double)(stop-start)/CLOCKS_PER_SEC;
printf("Posortowanie metoda babelkowa %d liczb zajmie: %lf s\n",n,czas);
```

Wynik działania powyższego fragmentu kodu.



```
"C:\Users\User\Desktop\Projekt algorytmy\Projekt\bin\Debug\Projekt.exe"
Ile liczb wylosowac: 10000
Posortowanie metoda babelkowa 10000 liczb zajmie: 0.313000 s

Process returned 0 (0x0)   execution time : 8.376 s
Press any key to continue.
```

Kolejny fragment kodu na początek pobiera od użytkownika ile liczb ma zostać wylosowanych i umieszczonych w tablicy. Następnie wypisuje zawartość tablicy za pomocą funkcji **print\_tab**. Kolejna funkcja **bubblesort** sortuje wypełnioną wcześniej tablicę i ponownie wypisuje jej zawartość.

```
int n;
printf("Ile liczb wylosowac: ");
scanf("%d", &n);

int tab[n];
fill(tab, n);
puts("Przed sortowaniem :");
print_tab(tab, n);
puts("");
puts("Po sortowaniu: ");
bubblesort(tab, n);
print_tab(tab, n);
```

Wynik działania tego fragmentu kodu wygląda następująco:



```
C:\Users\User\Desktop\Projekt algorytmy\Projekt\bin\Debug\Projekt.e
Ile liczb wylosowac: 10
Przed sortowaniem :
3 2 6 1 0 6 5 9 8 5
Po sortowaniu:
0 1 2 3 5 5 6 6 8 9
Process returned 0 (0x0)   execution time : 6.826 s
```

```

void swap(int *first, int *second)
{
    int temporary = *first;
    *first = *second;
    *second = temporary;
}

```

Funkcja ta zamienia miejscami pierwszy i ostatni element, jest ona używana zarówno do sortowania Heapsort jak i do Quciksort.

```

static inline int get_left_child_index(int index)
{
    return (index << 1) + 1;
}

```

Funkcja używana do Kopcowania (Heapsort), ma na celu zwrócenia indeksu lewego potomka.

```

static inline int get_right_child_index(int index)
{
    return (index << 1) + 2;
}

```

Analogicznie, tym razem funkcja zwraca indeks prawego potomka.

```

void heapify(int array[], int index, unsigned int size)
{
    int left = get_left_child_index(index),
        right = get_right_child_index(index),
        largest = index;
    if(left <= size)
        if(array[left] > array[index])
            largest = left;
    if(right <= size)
        if(array[right] > array[largest])
            largest = right;
    if(largest != index) {
        swap(&array[index], &array[largest]);
        heapify(array, largest, size);
    }
}

```

Podfunkcja sortowania Heapsort, ma na celu znalezienia największej wartości spośród trzech elementów (przodka, lewego potomka, prawego potomka) indeks elementu z największą wartością jest zapisywana do „largest” a na końcu porównywana z przodkiem, jeżeli wartość ta się zmieniła, podfunkcja wywoływana jest rekurencyjnie aby nie zaburzyć równowagi drzewa.

```

void build_heap(int array[], const int number_of_elements)
{
    int i;
    for(i = number_of_elements / 2; i >= 0; i--)
        heapify(array, i, number_of_elements - 1);
}

```

Funkcja buduje kopiec w tablicy która ma być posortowana.

```

void heapsort(int array[], int last_index)
{
    int i;

    build_heap(array, last_index);
    for(i=last_index; i>0; i--) {
        swap(&array[0], &array[i]);
        heapify(array, 0, --last_index);
    }
}

```

Główna funkcja kopcowania, przez parametr przesłany jest indeks ostatniego elementu oraz tablica.

```

int partition(int array[], int low, int high)
{
    int pivot = array[low];
    int i = low-1, j = high+1;

    while(i<j) {
        while(array[--j]>pivot)
            ;
        while(array[++i]<pivot)
            ;
        if(i<j)
            swap(&array[i], &array[j]);
    }
    return j;
}

```

Funkcja używana przez sortowanie quicksort aby znaleźć odpowiedni podział tablicy, na początku w pivocie zapisywany jest zawsze pierwszy element tablicy j ma wartość ostatniego elementu tablicy, while jest wykonywany dopóki idąc od końca nie napotkamy większego elementu od pivota, drugi while działa przeciwnie, jeśli i jest mniejsze od j z powodu minięcia, elementy są podmieniane.

```

void quicksort(int array[], int low, int high)
{
    if(low<high) {
        int partition_index = partition(array, low, high);
        quicksort(array, low, partition_index);
        quicksort(array, partition_index+1, high);
    }
}

```

Główna funkcja quicksort, partition dzieli tablice na 2 części, pierwsze wywołanie rekurencyjne jest wywoływane z 1 elementem tablicy jako początek i miejscem przecięcia na części pierwotnej tablicy, natomiast drugie wysyłanie rekurencyjne wywołuje się z pierwszym elementem o jednym dalej niż miejsce przecięcia i ostatnim elementem w tablicy.

```

void fill_zakres(int tab[], int n)
{
    srand(time(NULL));
    int i;

    int a,b,y,z;
    printf("Podaj zakres losowania liczb:\n");
    printf("Od: ");
    scanf("%d", &a);
    printf("Do: ");
    scanf("%d", &b);
    printf("Zakres %d %d\n", a,b);
    y=a;
    z=b;
    if(a<0){
        y=a*(-1);
    }
    if(b<0){
        z=b*(-1);
    }
    printf("y: %d z: %d\n", y,z);

    int x;
    if(a>=0){
        x=b-a+1;
    }
    else{
        x=y+z+1;
    }
    for(i=0;i<n;i++){
        tab[i]=(rand()%x)+a;
    }
    system("cls");
    printf("Zakres %d %d\n", a,b);
}

```

Funkcja pozwala nam wybrać zakres losowanych liczb, a następnie je losuje.

```

void print_tab(int tab[],int n)
{
    int i;
    for(i=0;i<n;i++){
        printf("%d ",tab[i]);
    }
}

```

Funkcja wypisująca elementy naszej tablicy.

```

void copy_tab(int tab[], int tab2[], int n)
{
    int i;
    for(i=0;i<n;i++){
        tab2[i]=tab[i];
    }
}

```

Funkcja która kopiuje elementy z jednej tablicy do drugiej.

```

void write_file(int tab[], FILE *f, int n)
{
    int i;
    for(i=0;i<n;i++){
        fprintf(f,"%d ",tab[i]);
    }
}

```

Funkcja zapisująca do pliku.

```

void read_file(int tab[], FILE *f, int n)
{
    int i;
    for(i=0;i<n;i++){
        fscanf(f, "%d", &tab[i]);
    }
}

```

Funkcja odczytująca z pliku.

```

int main()
{
    clock_t start, stop;
    double czas;
    int number;

    FILE *f;

    while(1){
        system("cls");
        printf("Co chcesz zrobic: \n\n");

        printf("1.Mierzenie czasu sortowania babelkowego dla n elementow\n");
        printf("2.Wyświetlenie tablicy przed i po sortowaniu\n");
        printf("3.Mierzenie czasu sortowania quicksort dla n elementow\n");
        printf("4.Sortowanie liczb podanych przez uzytkownika\n");
        printf("5.Porownanie czasu bubblesort, quicksort oraz heapsort\n");
        printf("6.Mierzenie czasu sortowania heapsort dla n elementow\n");
        printf("0.Zakonczenie dzialania programu \n");
        scanf("%d", &number);

        if(number==1){
            system("cls");

            int n;
            printf("Ile liczb wylosowac: ");
            scanf("%d", &n);

            int tab[n];
            fill_zakres(tab, n);

            start = clock();
            bubblesort(tab, n);
            stop = clock();
            czas = (double)(stop-start)/CLOCKS_PER_SEC;
            printf("Posortowanie metoda babelkowa %d liczb zajmie: %lf s\n", n, czas);
        }
    }
}

```

Główna funkcja main odpowiedzialna za wywołanie podfunkcji, oraz wyświetlająca menu wyboru. System(„cls”) czyści nam ekran np. przed wypisaniem na ekran poleceń do użytkownika. Start i stop = clock() jest używane do mierzenia sortowań z biblioteki time.h.

W mainie znajdują się również funkcje odpowiedzialne za poprawne działanie pliku oraz kontrola błędów.

## Działanie QuickSort



```

9 5 4 11 2 6 10
i           j

6 5 4 11 2 9 10
      i  j

6 5 4 2 11 9 10
      i  i

```

Działanie funkcji partition i jest ustawione na początku zawsze na pierwszym elemencie, j ma być mniejsze od naszego pivota czyli 9 a więc przez pętlę while znajduje 6, teraz te 2 liczby zamieniane są miejscami. Następnie i ma być większe od pivota a j mniejsze, znowu takie liczby są znajduwane i zamieniane miejscami, pętla jest wykonywana dopóki i i j się nie miną, jeśli tak się stanie zwracany jest j, który dzieli tablicę na 2 części.

```

6 4 3 2
i     j

2 5 4 6
      j  i

```

Pierwsza część teraz znowu jest dzielona na 2. Będziemy szli tylko lewą stroną.

```

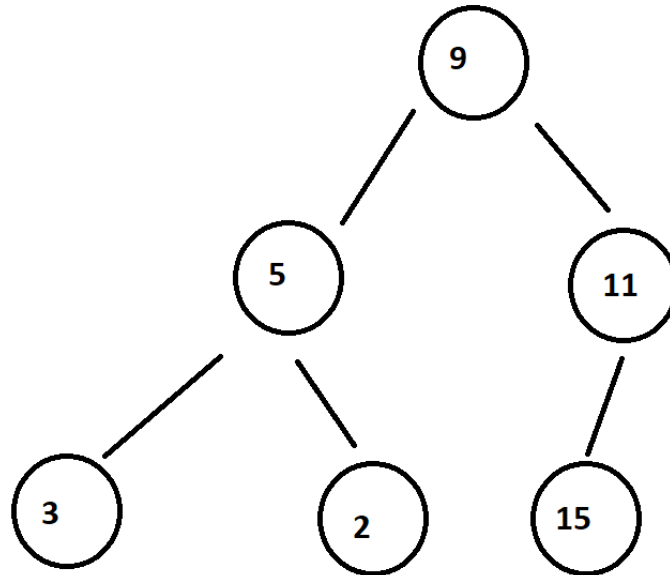
2 5 4
  j
  i

```

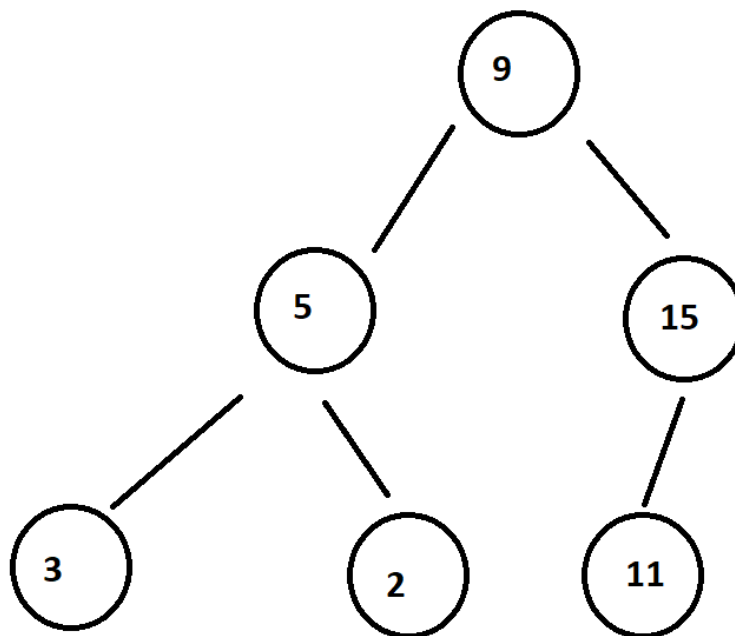
I jako pierwszy element do tablicy zapisywana jest 2 a 5 i 4 wysyłana jest jako druga część, tak działa sortowanie quicksort.

### Działanie Heapsort

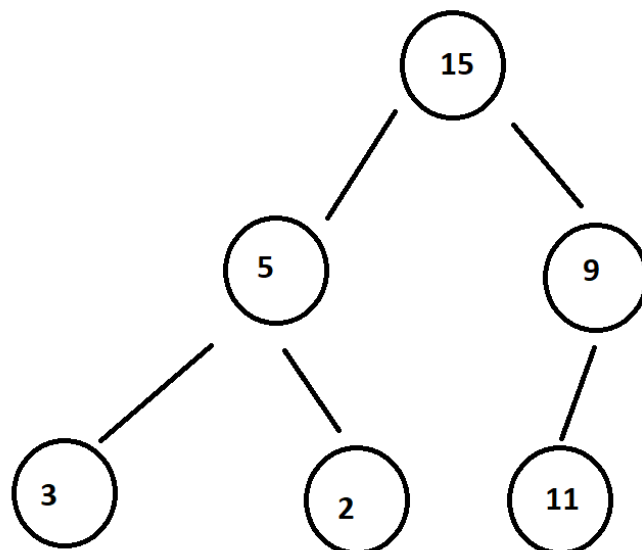
Kopcowanie może działać na 2 sposoby albo największy element jest zapisywany w korzeniu albo najmniejszy.



Na przykładzie będzie kopiec w którym korzeniu znajduje się największy element a więc pierwszym krokiem będzie czy w ostatnim rodzicu jest większa wartość niż w potomkach 11 jest mniejsze od 15 a więc zamieniamy te 2 liczby miejscami.

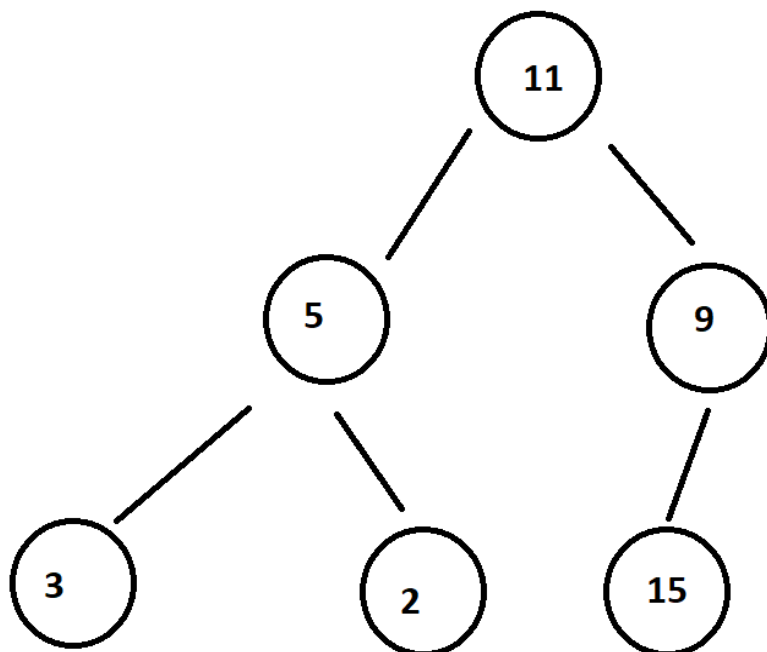


5 jest największa liczbą a więc zostawiamy tą gałąź bez zmian. Sprawdzamy teraz korzeń i jego potomków.

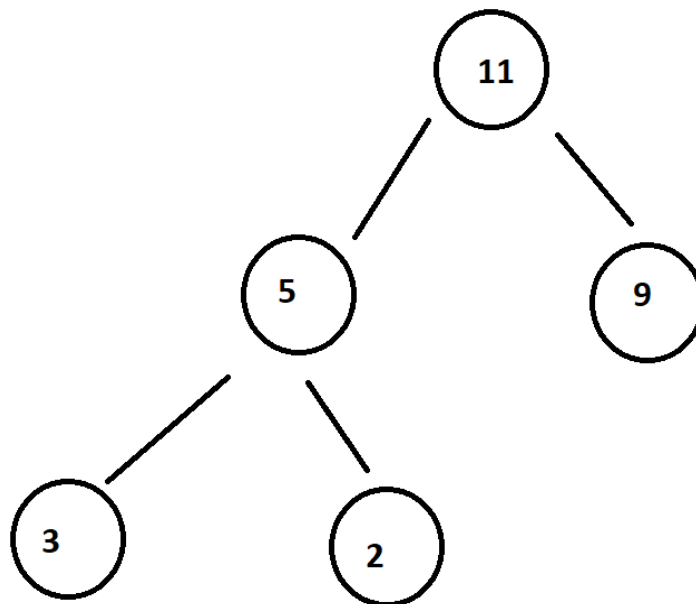


W prawym potomku była większa wartość a więc liczby miejscami, potem sprawdzamy rekurencyjnie znowu czy nie została zachwiana struktura prawego potomka poprzez zamienienie wartości.

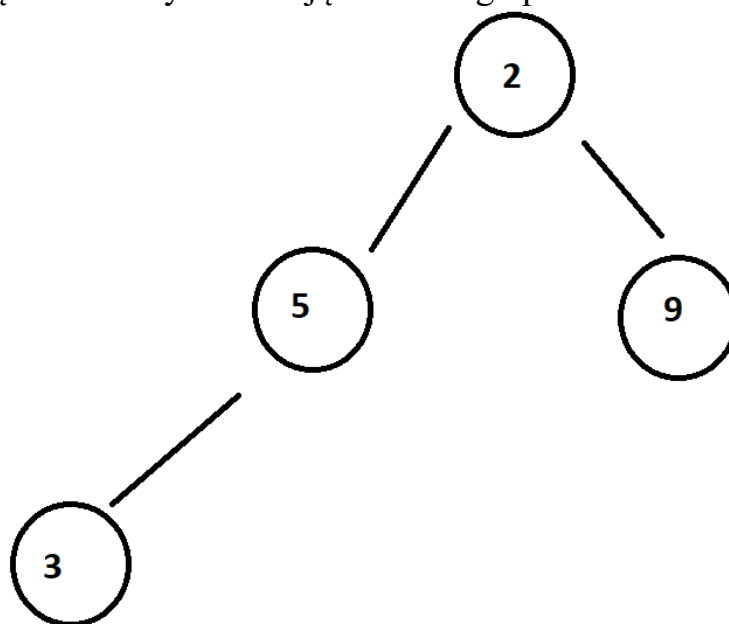
Mamy teraz pewność że w korzeniu znajduje się największa wartość. Zamieniamy miejscami z ostatnim miejscem i zapisujemy w tablicy.



Usuwamy ostatni liść z największym elementem, zmniejszając wielkość kopca.



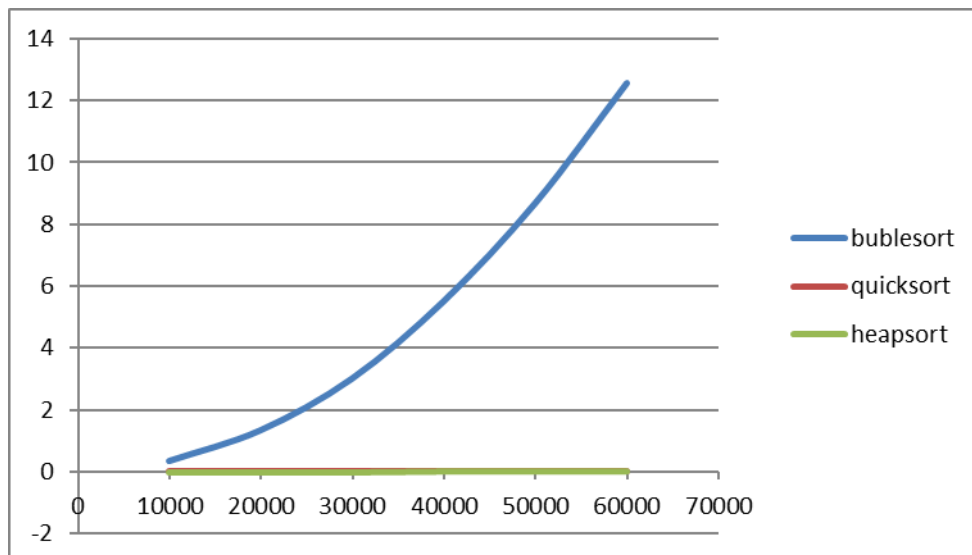
Sprawdzamy znowu czy przodkowie są więksi niż swoi potomkowie. W tym wypadku wszystko się zgadza, a więc program znowu zamienia miejscami 2 skrajne liczby, wstawiając do tablicy i usuwając ostatniego potomka.



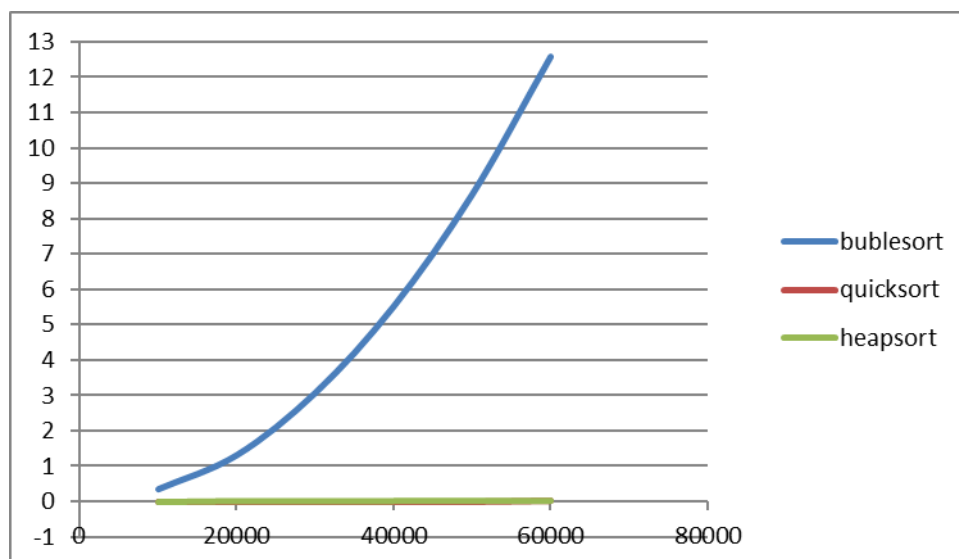
Analogicznie program działa tak dopóki nie zostanie ostatni element.

Mało efektywnym sortowaniem jest sortowanie bąbelkowe. Heapsort jest szybszy od bubblesorta, ale nie tak szybki jak quicksort,

zakres -1000 do 1000			
	bubblesort	quicksort	heapsort
10000	0,344	0	0
20000	1,343	0	0
30000	3,015	0	0
40000	5,514	0	0,015
50000	8,686	0	0,016
60000	12,562	0	0,016



zakres -100000 do 100000			
	bubblesort	quicksort	heapsort
10000	0,329	0	0
20000	1,297	0	0,015
30000	3,062	0	0,015
40000	5,514	0	0,016
50000	8,686	0	0,016
60000	12,591	0,016	0,021



zakres -500000 do 500000		
	quicksort	heapsort
100000	0,016	0,031
200000	0,031	0,063
300000	0,047	0,109
400000	0,062	0,141
500000	0,078	0,172

