Laboratorium 5

Szymon Kałuża

29.03.2022

Programy

BubbleSort

```
void bubbleSort(int* tab, int N){
    for(int i = 0; i < N - 1; i++)
    {
        for(int j = N - 1; j > i; j--)
        {
            if(tab[j] < tab[j-1])
            {
                change(tab, j, j-1);
            }
        }
    }
}
```

InsertSort

```
void insertSort(int* tab, int N){
    int temp = 0;
    int k = 0;
    for(int j = 1; j < N; j++){
        temp = tab[j];
        k = j - 1;
        while(k >= 0 && temp < tab[k]){
            tab[k + 1] = tab[k];
            k = k - 1;
        }
        tab[k + 1] = temp;
    }
}
```

SelectSort

```
void selectSort(int* tab, int N){
    for(int i = 0; i <= N -2; i++)
    {
        int min = i;
        for(int j = i + 1; j <= N -1; )
        {
            if(tab[j] < tab[min])
            {
                min = j;
            }
            j++;
        }
        change(tab, i, min);
    }
}
```

MinMax - DivideConcquer

```
int * minAndMaxDivideAndConquer(int* tab, int N){
    int min = 0;
    int max = 0;
    for(int i = 0; i < N - 2; i++){
        if(tab[i] < tab[i + 1]){
            if(tab[i] < tab[min]){
                min = i;
```

```
            }
            if(tab[i] > tab[max]){
                max = i;
            }
        }
        else{
            if(tab[i + 1] < tab[min]){
                min = i + 1;
            }
            if(tab[i] > tab[max]){
                max = i;
            }
        }
        i += 2;
    }
    if(N % 2 != 0){
        if(tab[N - 1] < tab[min]){
            min = N - 1;
        }
        if(tab[N - 1] > tab[max]){
            max = N - 1;
        }
    }
    static int score[2];
    score[0] = tab[min];
    score[1] = tab[max];
    return score;
}
```

MinMax - Naive

```
int* naiveMinMax(int* tab, int N){
    int min = 0;
    int max = 0;
    for(int j = 0; j < N; j++){
        if(tab[j] < tab[min]) min = j;
        if(tab[j] > tab[max]) max = j;
    }
    static int score[2];
    score[0] = tab[min];
    score[1] = tab[max];
    return score;
}
```

Exponentiation - DivideConcquer

```
long divideAndConquerExponentiation(long a, int n){
    long temp;
    if(n == 0){
        return 1;
    }
    else{
        if(n % 2 == 0){
            temp = divideAndConquerExponentiation(a, n/2);
            return temp * temp;
        }
        else{
            temp = divideAndConquerExponentiation(a, (n-1)/2);
            return temp * temp * a;
        }
    }
}
```

Exponentiation - Naive

```
int naiveExponentiation(int a, int n){
    int score = a;
    for(int j = 1; j < n; j++){
        score *= a;
    }
    return score;
}
```

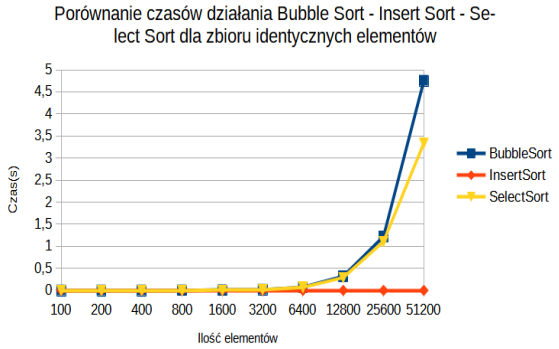
Zadanie1

Dane:

il.el	bubbleSort	selectSort	insertSort
100	4.4E-005	3.4E-005	2.8E-005
200	0.000164	0.00011	0.000102
400	0.000281	0.000174	0.00017
800	0.001123	0.000663	0.000673
1600	0.004378	0.002575	0.002713
3200	0.017429	0.010274	0.010848
6400	0.069817	0.040115	0.043356
12800	0.279136	0.157469	0.171953
25600	1.128569	0.628486	0.689997
51200	4.637079	2.510421	2.762975

Dane wynikające z symulacji wyraźnie wskazują na dominację sortowania przez wstawianie.

Figure 1: Porównanie Bubble Sort, Select Sort oraz Insert Sort dla tablicy jednakowych elementów

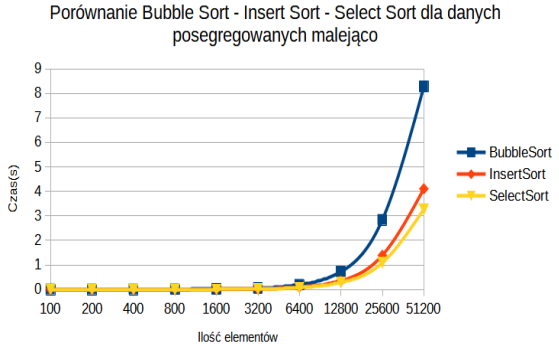


Zadanie 2

Dane:

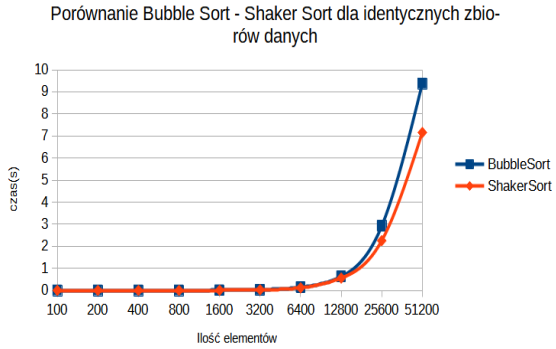
il. el	BubbleSort	InsertSort	SelectSort
100	6.1E-005	2.8E-005	2.7E-005
200	0.000241	0.000106	9.5E-005
400	0.000997	0.000439	0.000337
800	0.003867	0.001651	0.001281
1600	0.010621	0.005343	0.004686
3200	0.046712	0.022463	0.017782
6400	0.186684	0.089708	0.070575
12800	0.717315	0.349788	0.280748
25600	2.83609	1.388098	1.084036
51200	8.267241	4.102396	3.268428

Figure 2: Porównanie Bubble Sort, Select Sort oraz Insert Sort dla tablicy malejących elementów



Zadanie 3

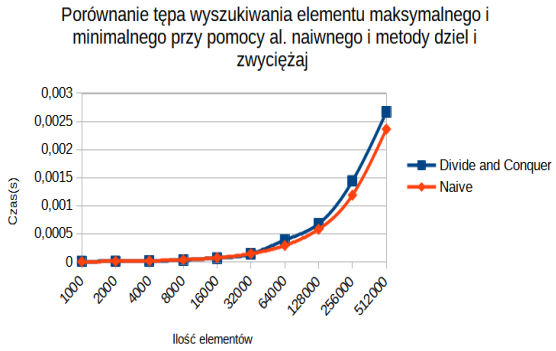
Figure 3: Porównanie Bubble Sort z Shaker Sort dla tablic losowych elementów



Zadanie 4

Zadanie 5

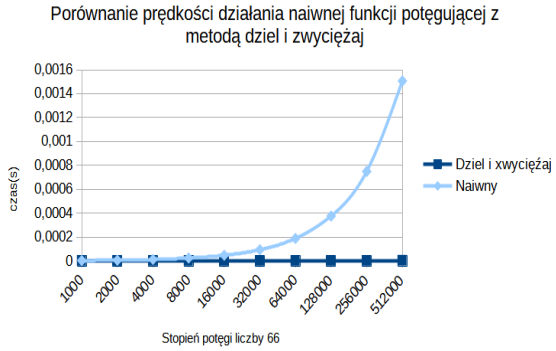
Figure 4: Porównanie szybkości działania naiwnego algorytmu wyszukiwania wartości maksymalnej i minimalnej z metodą dziel i zwyciężaj



Można zauważyć że szybkość działania tego programu jest nieintuicyjna. Jako że Divide conquer jest o wiele bardziej zaawansowanym algorytmem można dojść do wniosku że będzie działać szybciej, jednak prawda jest wręcz przeciwna. Divide conquer działa wolniej niż algorytm naiwny.

Zadanie 6

Figure 5: Porównanie szybkości działania naiwnego algorytmu potęgowania z metodą dziel i zwyciężaj



Można zauważyć, że algorytm divide conquer działa o wiele szybciej, nie zależnie od stopnia potęgi szybkość pozostaje praktycznie bez zmian, w momencie gdy w przypadku metody naiwnej prędkość rośnie w tempie geometryczny.