

Politechnika Warszawska	
Tytuł	Test algorytmu qsort.c
Autor	Monika Musielik
Sprawdzający	dr inż. Piotr Witkowski
Data	8 listopada 2025

Spis treści

1	Cel projektu	1
2	Sortowanie przez wstawianie	1
3	Szybkie sortowanie	2

1 Cel projektu

Celem projektu było odnalezienie i naprawa błędów w algorytmach sortowania przez wstawianie oraz szybkiego sortowania, a następnie udokumentowanie tego procesu w sprawozdaniu.

2 Sortowanie przez wstawianie

W pliku `insort.c` została napisana błędna funkcja algorytmu sortowania przez wstawianie, po niewielkim sprzątnięciu prezentuje się ta funkcja w sposób następujący:

```

3 void insort(double v[], int n) {
4     int i, j;
5     for (i = 1; i < n; i++) {
6         double tmp = v[j];
7         for (j = i-1; v[j] > tmp, j--)
8             v[j+1] = v[j];
9         v[j+1] = tmp;
10    }
11 }
```

Plik niestety się nie kompiluje ze względu na błąd składni w liniach 7 i 9. W linii 7 został zamieniony średnik z przecinkiem, a w linii 9 litera `n` została zamieniona z literą `m`.

Po szybkich naprawach i uruchomieniu programu, można zauważyć, że nie działa on tak jak powinien i nie sortuje tablicy zgodnie z zasadami matematyki, a czasem występuje błąd segmentacji.

Od razu można było zauważyć błąd w linii 6, używanie zmiennej `j`, zanim została jej przypisana jakakolwiek wartość. Jest to kolejna literówka, ponieważ do tej zmiennej powinna być przypisywana wartość `v[i]`, zamiast `v[j]`.

Program teraz o wiele częściej zwraca poprawnie posortowany wektor, natomiast nadal występują błędy segmentacji, co sugeruje naruszanie ochrony pamięci, czyli brak warunku w którejś z pętli. Pętla `for` w linii 5 nie powinna *mazać tam, gdzie nie powinna*, ponieważ ma jasno określony zakres operacji. Natomiast pętla w linii 7 już nie ma `i` i może zejść poniżej `j = 0`, dlatego należy dodać warunek który to uniemożliwia.

```

3 void insort(double v[], int n) {
4     int i, j;
5     for (i = 1; i < n; i++) {
6         double tmp = v[i];
7         for (j = i-1; j >= 0 && v[j] > tmp; j--)
8             v[j+1] = v[j];
9         v[j+1] = tmp;
10    }
11 }

```

Po powyższych poprawkach funkcja działa zgodnie z założeniami i zwraca poprawne wyniki. Błędy były proste do zlokalizowania i naprawienia głównie przez prostotę algorytmu sortowania przez wstawianie.

3 Szybkie sortowanie

W pliku `q_sort.c` znajdują się 3 funkcje: `q_sort`, `qsort_rec`, `divide`. Pierwsze 2 funkcje nie mają błędów, można to łatwo zbadać przez ich prostotę:

```

33 void qsort_rec( double v[], int first, int last ) {
34     if( first < last ) {
35         int m = divide( v, first, last );
36         qsort_rec( v, first, m-1 );
37         qsort_rec( v, m+1, last );
38     }
39 }
40
41 void q_sort( double v[], int n ) {
42     qsort_rec( v, 0, n-1 );
43 }

```

Funkcja `q_sort` opakowuje funkcje `qsort_rec` ze zmniejszoną wartością n o 1. Natomiast główna funkcja `qsort_rec`, jest rekurencyjna i wywołuje funkcje `divide` która dzieli tablicę.

Program kompiluje się bez zmian, jednak zachodzą w nim błędy przy porównywaniu liczb. Dzięki temu można dojść do wniosku, że błąd pojawia się w funkcji `divide`. Dla skrócenia funkcji dodam prostą funkcję `swap` zamieniającą między sobą 2 elementy.

```

3 void swap(double *a, double *b) {
4     double tmp;
5     tmp = *a;
6     *a = *b;
7     *b = tmp;
8 }

```

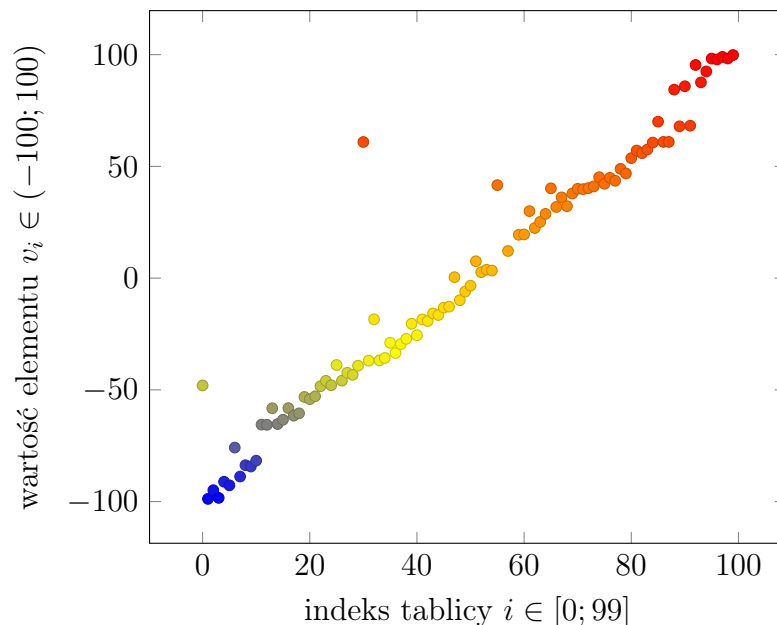
Dzięki tej prostej zmianie funkcja dzielenia tablic znacząco poprawiła swoją czytelność i jest prosta w analizowaniu błędów, który nadal się nie zmienił:

```

10 int divide( double v[], int f, int l ) {
11     int s = f;
12     f++;
13
14     while( f < l ) {
15         while( f < l && v[f] < v[s] ) f++;
16         while( f < l && v[l] > v[s] ) l--;
17         if( f < l ) swap(&v[f], &v[l]);
18     }
19
20     swap(&v[s], &v[f]);
21     return f;
22 }

```

Po uruchomieniu programu można zauważyć ciekawą rzecz, wektor po przejściu przez tę funkcję nie jest posortowany, jednak po wyświetleniu przykładowej tablicy złożonej z liczb pseudolosowych o $n = 100$ z $v_n \in (-100; 100)$ przeprowadzoną przez funkcję `q_sort` można zobaczyć specyficzny kształt danych na wykresie punktowym:



Jak widać tablica jest *prawie* posortowana. Postanowiłam usunąć z wykresu 2 elementy: $v_{56} = 21094$, $v_{58} = 44531$, ponieważ ich wartości sięgają dziesiątki tysięcy, oraz nie posiadają części ułamkowej, co sugeruje naruszenie ochrony pamięci.

Innym prostym błędem jest błędna końcowa zamiana elementu startowego v_f z elementem piwotu v_s . Zgodnie z algorytmem szybkiego sortowania element piwotu powinien być zamieniony z *ostatnim elementem wektora*, tak wygląda poprawiona wersja:

```

20 swap(&v[s], &v[l]);
21 return l;

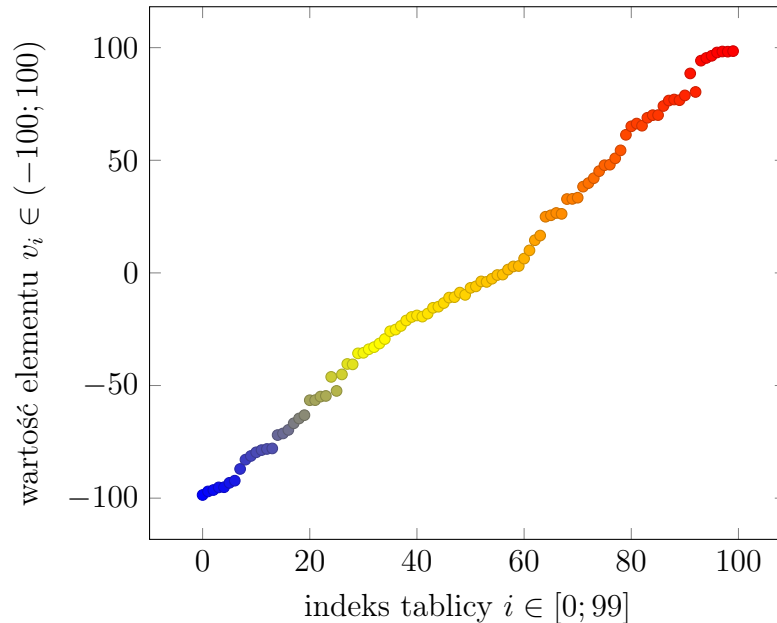
```

Można zauważyć na wykresie wyżej inny problem, pojedyncze wartości są bardzo odchylone od siebie. Jest to spowodowane błędami warunków w liniach 15 i 16. Można je poprawić zamieniając operator `<` na `<=`

```

15 while( f <= l && v[f] < v[l] ) f++;
16 while( f <= l && v[l] > v[f] ) l--;

```



Jak widać ta zmiana bardzo poprawiła algorytm sortowania, który jest coraz bardziej zgodny z założeniami. Nie występują już pojedyncze punkty które są po przeciwnych stronach sekcji. Jednak algorytm ciągle ma błędy.

Kolejnym błędem jest błędny warunek zakończenia głównej pętli `while`, ponownie jak w błędzie wyżej należy zamienić operator `<` na `<=`.

```

14 while( f <= l ) {

```

Po tej zmianie wydawałoby się, że algorytm już działa poprawnie, ale jeśli w pliku `test.c` wymuszę aby 2 elementy miały dokładnie taką samą wartość, np. $v_5 = 10$, $v_6 = 10$ to program wejdzie w nieskończoną pętlę.

Aby naprawić ten błąd należy rozszerzyć kod w instrukcji warunkowej, tak aby przesuwał wskaźniki f i l oraz kończył w odpowiednim momencie pętlę:

```

17 if( f < l ) {
18     swap(&v[f], &v[l]);
19     f++;
20     l--;
21 } else break;

```

Po wszystkich poprawkach algorytm sortowania działa zgodnie z założeniami i zwraca poprawnie wyniki. W porównaniu do sortowania przez wstawianie, naprawa tego algorytmu była znacznie trudniejsza ze względu na większą złożoność kodu który po wszystkich naprawach wygląda w sposób następujący:

```

3 void swap(double *a, double *b) {
4     double tmp;
5     tmp = *a;
6     *a = *b;
7     *b = tmp;
8 }
9
10 int divide( double v[], int f, int l ) {
11     int s = f;
12     f++;
13
14     while( f <= l ) {
15         while( f <= l && v[f] < v[s] ) f++;
16         while( f <= l && v[l] > v[s] ) l--;
17         if( f < l ) {
18             swap(&v[f], &v[l]);
19             f++;
20             l--;
21         } else break;
22     }
23
24     swap(&v[s], &v[l]);
25     return l;
26 }
27
28 void qsort_rec( double v[], int first, int last ) {
29     if( first < last ) {
30         int m = divide( v, first, last );
31         qsort_rec( v, first, m-1 );
32         qsort_rec( v, m+1, last );
33     }
34 }
35
36 void q_sort( double v[], int n ) {
37     qsort_rec( v, 0, n-1 );
38 }

```