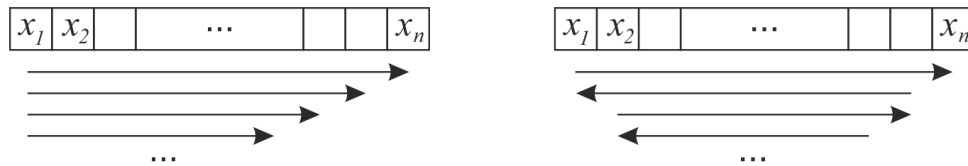


1. Zrealizować procedurę sortowania przez zamianę `void sort1(int x[], int n)`, która w k -tym przebiegu wyszukuje najmniejszy wśród elementów $x[k+1], \dots, x[n]$ i zamienia go z $x[k]$.
2. Napisać procedurę sortowania przez wstawianie `void sort2(int x[], int n)`, która w k -tym przebiegu przesuwając element $x[k]$ w lewo o odpowiednią liczbę miejsc wśród już posortowanych elementów $x[0], \dots, x[k-1]$. Algorytm ten przypomina popularny wśród brydżystów sposób porządkowania kart w ręku bezpośrednio po rozdaniu.
3. Zmodyfikować algorytm z zad. 2 w taki sposób, aby wyznaczenie miejsca do przestawienia elementu $x[k]$ odbywało się przez połówkowe przeszukiwanie ciągu $x[0], \dots, x[k-1]$.
4. Napisać procedurę `void sort3(int x[], int n)` sortującą dane w tablicy x metodą bąbelkową. Następnie zaproponować ulepszenia podstawowej metody: a) wykorzystujące zapamiętaną w poprzednim przebiegu pozycję k ostatniej wykonanej zamiany $x[k] \leftrightarrow x[k+1]$; b) wykorzystujące zapamiętaną w poprzednim przebiegu pozycję pierwszej zamiany $x[l] \leftrightarrow x[l+1]$; c) zmieniającej w kolejnych przebiegach kierunek przeglądania tablicy (por. rys. poniżej); d) połączenie ulepszeń a), b) i c). Przeanalizować w jaki sposób zaproponowane ulepszenia wpływają na wykonywaną liczbę porównań i zamian elementów $x[i]$ i $x[i+1]$ w stosunku do rozwiązania podstawowego.



5. Metoda Shella jest modyfikacją algorytmu sortowania bąbelkowego, polegającą na wstępnym posortowaniu elementów tablicy odległych o $h > 1$, a więc $x[0], x[h], x[2h], \dots$, dalej $x[1], x[h+1], x[2h+1], \dots$ itd. Kolejny przebieg działa podobnie, lecz dla mniejszej wartości h . W ostatnim przebiegu $h=1$ i algorytm działa tak, jak zwykle sortowanie bąbelkowe. Zysk jaki uzyskuje się w metodzie Shella w stosunku do podstawowego algorytmu polega na tym, że we wstępnych przebiegach tablica ulega częściowemu posortowaniu znacznie mniejszym nakładem pracy, gdyż duże i małe elementy przemieszczają się na koniec i odp. na początek tablicy w niewielkiej liczbie zamian wykonywanych nie między bezpośrednimi sąsiadami, lecz między elementami w znacznej odległości h . Doświadczalnie zbadano jakie sekwencje odległości $h_k \geq h_{k-1} \geq \dots \geq h_0 = 1$ dają najlepsze wyniki. Jedną z proponowanych sekwencji to liczby spełniające zależność $h_j = (3^{j+1} - 1)/2$, a więc 1, 4, 13, 40 itd., przy czym największą z nich wybiera się tak, aby nie przekraczała ona jednej trzeciej rozmiaru tablicy x . Napisać program realizujący sortowanie metodą Shella.
6. Zrealizować sortowanie przez kopcowanie. Kopiec tworzony jest w tablicy.
7. Eksperyment – opracować statystyki dla wybranych metod sortowania: bąbelkowe, wybór, wstawianie, Shell, kopiec, quicksort. Należy posortować ciąg 10000 losowo wygenerowanych liczb każdą z ocenianych metod, rejestrując przy tym liczbę wykonanych porównań i podstawień sortowanych wielkości. Sortowanie należy powtórzyć 100–200 razy, za każdym razem generując nowe dane, i na tej podstawie wyznaczyć maksymalne, minimalne oraz średnie liczby porównań i podstawień dla każdej z metod. Statystykę uzupełnić analizą zachowania algorytmów na danych dokładnie posortowanych lub posortowanych odwrotnie. Szczegółowe omówienie – na ćwiczeniach.
8. Zrealizować kolejkę priorytetową z wykorzystaniem kopca. Potrzebne są standardowe operacje: `push(Q,k)` – dodanie nowego elementu k do kolejki, `pop(Q)` – pobranie z kolejki elementu o maksymalnym priorytecie, `empty(Q)` – testowanie pustości kolejki. Prócz tego (zad. 13) potrzebna będzie funkcja `update(Q,k,p)` zmieniająca priorytet wskazanego elementu k na wartość p i odpowiednio przesuwająca go w kolejce. Należy użyć pomocniczej tablicy `Index[k]`, która wskazuje aktualną pozycję elementu k w kolejce, by uniknąć w ten sposób czasochłonnego jej przeszukiwania.
9. Implementacja algorytmu Kruskala budowy optymalnego drzewa spinającego grafu z wykorzystaniem kolejki z zad. 8.
10. Napisać procedurę, która czyta z konsoli pary dodatnich liczb `int` aż do wpisania (0,0), interpretując je jako opisy krawędzi grafu a) skierowanego lub b) nieskierowanego. Zadaniem tej procedury jest utworzenie reprezentacji grafu w postaci list sąsiedztwa.
11. Zaimplementować algorytmy DFS i BFS wykorzystujące listy sąsiedztwa utworzone w zad. 10. Powinny one wyprowadzać ciągi wierzchołków grafu w kolejności ich odwiedzenia.
12. Dla zadanego grafu nieskierowanego należy utworzyć bazę cykli prostych wykorzystując modyfikację algorytmu DFS.
13. Zrealizować algorytm Dijkstry wyszukiwania najkrótszych ścieżek w grafie skierowanym lub nieskierowanym z zadanego wierzchołka s wykorzystujący funkcje obsługi kolejki priorytetowej, która przechowuje wierzchołki grafu z priorytetami równymi najmniejszym znalezionym do tej pory odległościom poszczególnych wierzchołków od źródła s .
14. Zrealizować algorytm Floyda-Warshalla wyszukiwania najkrótszych dróg w grafie skierowanym lub nieskierowanym zadanym przez macierz odległości sąsiadów. Zmodyfikować następnie algorytm tak, aby wyznaczał on tranzytywne domknięcie grafu skierowanego.