SelectionSort, InsertionSort, CoctailSort, HeapSort, QuickSort – porównanie działania algorytmów

# Kod źródłowy

## InsertionSort

private void InsertionSort(int[] Tab)

{

for (uint i = 1; i < Tab.Length; i++)

{

uint j = i;

int temp = Tab[j];

while ((j > 0) && (Tab[j - 1] > temp))

{

Tab[j] = Tab[j - 1];

j--;

}

Tab[j] = temp;

}

}

## SelectionSort

private void SelectionSort(int[] Tab)

{

uint k;

for (uint i = 0; i < (Tab.Length - 1); i++)

{

int temp = Tab[i];

k = i;

for (uint j = i + 1; j < Tab.Length; j++)

{

if (Tab[j] < temp)

{

k = j;

temp = Tab[j];

}

}

Tab[k] = Tab[i];

Tab[i] = temp;

}

}

## HeapSort

private void Heapify(int[] Tab, int Left, int Right)

{

int i = Left,

j = 2 \* i + 1;

int temp = Tab[i];

while (j <= Right)

{

if (j < Right)

{

if (Tab[j] < Tab[j + 1])

{

j++;

}

}

if (temp >= Tab[j])

{

break;

}

Tab[i] = Tab[j];

i = j;

j = 2 \* i + 1;

## CoctailSort

private void CocktailSort(int[] Tab)

{

int left = 1;

int right = Tab.Length - 1;

int k = Tab.Length - 1;

do

{

for (int j = right; j >= left; j--)

{

if (Tab[j - 1] > Tab[j])

{

int temp = Tab[j - 1];

Tab[j - 1] = Tab[j];

Tab[j] = temp;

k = j;

}

}

left = k + 1;

for (int j = left; j <= right; j++)

{

if (Tab[j - 1] > Tab[j])

{

int temp = Tab[j - 1];

Tab[j - 1] = Tab[j];

Tab[j] = temp;

k = j;

}

}

right = k - 1;

}

while (left <= right);

}

}

Tab[i] = temp;

}

private void HeapSort(int[] Tab)

{

Heapify(Tab, 0, Tab.Length - 1);

int left = Tab.Length / 2,

right = Tab.Length - 1;

while (left > 0)

{

left--;

Heapify(Tab, left, right);

}

while (right > 0)

{

int temp = Tab[left];

Tab[left] = Tab[right];

Tab[right] = temp;

right--;

Heapify(Tab, left, right);

}

}

QuickSort rekurencyjny

private void QuickSortRecursive(int[] Tab, int Left, int Right, int Key)

{

int i;

int j;

int x;

i = Left;

j = Right;

if (Key == 0) x = Tab[Left]; // Left

else if (Key == 1) x = Tab[Right]; // Right

else if (Key == 2) x = Tab[(Left + Right) / 2]; // Middle

else x = Tab[Left + random.Next(Right - Left)]; // Random

do

{

while (Tab[i] < x)

{

i++;

}

while (x < Tab[j])

{

j--;

}

if (i <= j)

{

int temp = Tab[i];

Tab[i] = Tab[j];

Tab[j] = temp;

i++;

j--;

}

}

while (i <= j);

if (Left < j)

{

QuickSortRecursive(Tab, Left, j, Key);

}

if (i < Right)

{

QuickSortRecursive(Tab, i, Right, Key);

}

}

QuickSort (iteracyjny)

private void QuickSortIterative(int[] Tab, int Key)

{

int i, j;

int Left, Right, sp;

int[] StackLeft = new int[Tab.Length];

int[] StackRight = new int[Tab.Length];

sp = 0;

StackLeft[sp] = 0;

StackRight[sp] = Tab.Length - 1;

do

{

Left = StackLeft[sp];

Right = StackRight[sp];

sp--;

do

{

int x;

i = Left;

j = Right;

if (Key == 0) x = Tab[Left]; // Left

else if (Key == 1) x = Tab[Right]; // Right

else if (Key == 2) x = Tab[(Left + Right) / 2]; // Middle

else x = Tab[Left + random.Next(Right - Left)]; // Random

do

{

while (Tab[i] < x) i++;

while (x < Tab[j]) j--;

if (i <= j)

{

int t = Tab[i];

Tab[i] = Tab[j];

Tab[j] = t;

i++; j--;

}

} while (i <= j);

if (i < Right)

{

sp++;

StackLeft[sp] = i;

StackRight[sp] = Right;

}

Right = j;

} while (Left < Right);

} while (sp >= 0);

}

# Porównaj szybkość działania 4 metod sortowania: Insertion Sort, Selection Sort, Heap Sort, Cocktail Sort dla tablicy  liczb całkowitych (rzędu 50k ‐ 200k elementów) generowanych w postaci: losowej, rosnącej, malejącej, stałej, v-kształtnej.

## Dane losowe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dane losowe | |  |
| N | InsertionSort | SelectionSort | CoctailSort | HeapSort |
| 50000 | 39.89 | 97.28 | 217.36 | 0.72 |
| 60000 | 41.12 | 137.63 | 284.48 | 0.94 |
| 70000 | 55.89 | 214.90 | 505.19 | 3.77 |
| 80000 | 128.95 | 268.50 | 521.91 | 0.93 |
| 90000 | 86.92 | 362.25 | 623.52 | 0.86 |
| 100000 | 114.96 | 482.87 | 830.79 | 0.84 |
| 110000 | 122.77 | 452.91 | 1085.24 | 0.91 |
| 120000 | 146.83 | 537.47 | 944.82 | 1.08 |
| 130000 | 187.33 | 666.33 | 1326.77 | 1.22 |
| 140000 | 224.30 | 757.62 | 1461.51 | 1.33 |
| 150000 | 220.80 | 977.55 | 2135.15 | 6.13 |
| 160000 | 293.50 | 923.40 | 1661.88 | 1.49 |
| 170000 | 257.37 | 1032.61 | 1802.82 | 1.71 |
| 180000 | 299.38 | 1036.39 | 2116.33 | 1.66 |
| 190000 | 352.10 | 1108.26 | 2516.88 | 1.89 |
| 200000 | 375.58 | 1423.06 | 2866.01 | 1.90 |

## Dane malejące

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dane malejące | |  |
| N | InsertionSort | SelectionSort | CoctailSort | HeapSort |
| 50000 | 51.99 | 85.06 | 98.38 | 0.31 |
| 60000 | 71.80 | 129.78 | 160.43 | 0.35 |
| 70000 | 89.10 | 164.60 | 194.14 | 0.41 |
| 80000 | 131.74 | 218.48 | 259.56 | 0.43 |
| 90000 | 149.44 | 292.69 | 339.56 | 0.52 |
| 100000 | 190.03 | 311.43 | 460.25 | 0.54 |
| 110000 | 250.92 | 366.19 | 505.21 | 0.59 |
| 120000 | 269.97 | 484.36 | 674.14 | 0.65 |
| 130000 | 339.39 | 505.18 | 782.47 | 0.76 |
| 140000 | 371.84 | 695.93 | 793.75 | 0.78 |
| 150000 | 425.49 | 684.56 | 912.87 | 0.91 |
| 160000 | 496.57 | 829.23 | 1056.09 | 0.91 |
| 170000 | 585.71 | 946.45 | 1169.78 | 0.94 |
| 180000 | 576.10 | 1113.53 | 1371.40 | 2.06 |
| 190000 | 681.11 | 1211.54 | 1410.29 | 1.02 |
| 200000 | 768.62 | 1214.66 | 1607.44 | 1.24 |

## Dane rosnące

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dane rosnące | |  |
| N | InsertionSort | SelectionSort | CoctailSort | HeapSort |
| 50000 | 0.01 | 76.38 | 0.00 | 0.28 |
| 60000 | 0.01 | 115.02 | 0.01 | 0.30 |
| 70000 | 0.01 | 168.27 | 0.00 | 0.35 |
| 80000 | 0.02 | 210.77 | 0.01 | 0.42 |
| 90000 | 0.02 | 284.26 | 0.01 | 0.44 |
| 100000 | 0.02 | 320.15 | 0.01 | 0.60 |
| 110000 | 0.02 | 409.06 | 0.01 | 0.53 |
| 120000 | 0.02 | 467.97 | 0.01 | 0.66 |
| 130000 | 0.03 | 562.20 | 0.01 | 0.69 |
| 140000 | 0.03 | 666.56 | 0.01 | 0.75 |
| 150000 | 0.03 | 741.12 | 0.02 | 0.75 |
| 160000 | 0.03 | 810.50 | 0.01 | 0.87 |
| 170000 | 0.03 | 1069.73 | 0.01 | 0.86 |
| 180000 | 0.04 | 1125.69 | 0.01 | 0.97 |
| 190000 | 0.04 | 1097.15 | 0.02 | 1.12 |
| 200000 | 0.04 | 1208.48 | 0.03 | 1.14 |

## Dane stałe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dane stałe | |  |
| N | InsertionSort | SelectionSort | CoctailSort | HeapSort |
| 50000 | 0.01 | 75.42 | 0.00 | 0.04 |
| 60000 | 0.01 | 124.92 | 0.00 | 0.04 |
| 70000 | 0.02 | 192.49 | 0.01 | 0.05 |
| 80000 | 0.02 | 232.66 | 0.01 | 0.05 |
| 90000 | 0.02 | 323.42 | 0.01 | 0.06 |
| 100000 | 0.02 | 399.05 | 0.01 | 0.06 |
| 110000 | 0.02 | 452.59 | 0.01 | 0.07 |
| 120000 | 0.02 | 482.05 | 0.01 | 0.07 |
| 130000 | 0.03 | 536.37 | 0.01 | 0.10 |
| 140000 | 0.03 | 866.00 | 0.02 | 0.15 |
| 150000 | 0.03 | 938.73 | 0.02 | 0.10 |
| 160000 | 0.05 | 995.60 | 0.02 | 0.10 |
| 170000 | 0.04 | 1196.80 | 0.01 | 0.12 |
| 180000 | 0.04 | 1351.17 | 0.02 | 0.12 |
| 190000 | 0.04 | 1157.71 | 0.02 | 0.13 |
| 200000 | 0.04 | 1479.67 | 0.02 | 0.12 |

## Dane V-kształtne

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Dane V-kształtne | |  |
| N | InsertionSort | SelectionSort | CoctailSort | HeapSort |
| 50000 | 27.84 | 107.72 | 200.63 | 0.89 |
| 60000 | 48.70 | 130.06 | 260.11 | 1.81 |
| 70000 | 54.50 | 204.87 | 405.40 | 0.93 |
| 80000 | 103.77 | 315.91 | 590.06 | 0.79 |
| 90000 | 179.12 | 365.22 | 626.69 | 1.14 |
| 100000 | 130.85 | 391.44 | 819.92 | 1.02 |
| 110000 | 135.66 | 500.22 | 879.92 | 1.17 |
| 120000 | 152.83 | 580.33 | 1045.25 | 1.16 |
| 130000 | 165.53 | 605.85 | 1225.95 | 1.32 |
| 140000 | 223.29 | 681.31 | 1578.67 | 1.40 |
| 150000 | 262.19 | 904.86 | 1826.19 | 1.45 |
| 160000 | 260.38 | 936.98 | 1700.63 | 1.55 |
| 170000 | 288.79 | 1046.37 | 2186.36 | 1.62 |
| 180000 | 371.07 | 1380.01 | 2320.68 | 2.47 |
| 190000 | 364.17 | 1270.30 | 2898.65 | 2.02 |
| 200000 | 501.92 | 1537.27 | 3584.08 | 2.12 |

# Dla różnych typów danych wejściowych porównaj efektywność działania powyższych algorytmów.  Przedstaw wykresy t = f(n) dla każdego typu danych wejściowych i różnych metod sortowania (5 wykresów).  Liczbę elementów należy dobrać w taki sposób, aby możliwe było wykonanie pomiarów.

## InsertionSort

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | InsertionSort | |  |
| N | Losowe | Malejące | Rosnące | Stałe | V |
| 50000 | 42.94 | 53.97 | 0.01 | 0.01 | 43.79 |
| 60000 | 32.72 | 64.21 | 0.01 | 0.01 | 31.04 |
| 70000 | 42.46 | 84.18 | 0.06 | 0.01 | 37.15 |
| 80000 | 50.91 | 114.00 | 0.01 | 0.02 | 50.75 |
| 90000 | 61.83 | 136.12 | 0.01 | 0.02 | 62.56 |
| 100000 | 83.12 | 164.62 | 0.02 | 0.02 | 73.50 |
| 110000 | 96.87 | 191.74 | 0.02 | 0.02 | 99.24 |
| 120000 | 110.09 | 251.14 | 0.02 | 0.02 | 130.96 |
| 130000 | 160.17 | 268.07 | 0.03 | 0.06 | 132.71 |
| 140000 | 144.58 | 354.54 | 0.02 | 0.02 | 175.60 |
| 150000 | 178.93 | 373.97 | 0.02 | 0.02 | 153.46 |
| 160000 | 247.17 | 400.99 | 0.03 | 0.03 | 215.29 |
| 170000 | 307.03 | 431.69 | 0.03 | 0.03 | 220.41 |
| 180000 | 292.48 | 565.24 | 0.03 | 0.03 | 292.22 |
| 190000 | 308.34 | 717.32 | 0.03 | 0.04 | 305.41 |
| 200000 | 366.77 | 613.54 | 0.03 | 0.03 | 270.34 |

## SelectionSort

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | SelectionSort | |  |
| N | Losowe | Malejące | Rosnące | Stałe | V |
| 50000 | 64.72 | 114.19 | 64.91 | 68.60 | 82.53 |
| 60000 | 101.42 | 110.88 | 94.12 | 97.04 | 88.32 |
| 70000 | 130.97 | 116.67 | 154.21 | 124.41 | 124.80 |
| 80000 | 172.72 | 251.91 | 221.34 | 178.27 | 159.19 |
| 90000 | 224.17 | 219.42 | 241.75 | 207.13 | 194.32 |
| 100000 | 263.86 | 283.03 | 264.72 | 287.53 | 247.58 |
| 110000 | 303.38 | 286.19 | 397.23 | 346.17 | 321.39 |
| 120000 | 392.75 | 381.52 | 417.42 | 353.00 | 341.61 |
| 130000 | 497.98 | 573.03 | 466.92 | 429.48 | 399.02 |
| 140000 | 468.69 | 579.45 | 523.81 | 561.09 | 546.85 |
| 150000 | 719.81 | 596.52 | 742.11 | 568.26 | 560.66 |
| 160000 | 733.34 | 606.53 | 768.88 | 626.42 | 590.85 |
| 170000 | 1095.20 | 769.36 | 935.80 | 737.99 | 695.52 |
| 180000 | 929.72 | 969.24 | 942.77 | 845.60 | 994.18 |
| 190000 | 1002.55 | 1120.95 | 912.86 | 990.21 | 1007.97 |
| 200000 | 1105.40 | 1047.05 | 1023.81 | 1001.02 | 1182.08 |

## CoctailSort

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | CoctailSort | |  |
| N | Losowe | Malejące | Rosnące | Stałe | V |
| 50000 | 136.1439 | 106.4871 | 0.003472 | 0.003383 | 173.3271 |
| 60000 | 186.5168 | 106.9205 | 0.003301 | 0.003411 | 195.513 |
| 70000 | 249.3262 | 164.0645 | 0.004791 | 0.004608 | 261.1047 |
| 80000 | 306.1706 | 395.6842 | 0.005099 | 0.004588 | 326.5584 |
| 90000 | 465.0239 | 319.6066 | 0.00681 | 0.004695 | 422.9297 |
| 100000 | 527.8207 | 314.8005 | 0.007196 | 0.007084 | 639.581 |
| 110000 | 614.5683 | 468.5752 | 0.007001 | 0.006533 | 669.7459 |
| 120000 | 741.3639 | 580.9437 | 0.008404 | 0.008056 | 806.9788 |
| 130000 | 831.2913 | 727.8602 | 0.007589 | 0.008713 | 919.4729 |
| 140000 | 1094.364 | 742.6841 | 0.009542 | 0.00882 | 1118.38 |
| 150000 | 1415.659 | 878.2548 | 0.0101 | 0.01 | 1249.407 |
| 160000 | 1790.814 | 927.5932 | 0.009522 | 0.010297 | 1578.492 |
| 170000 | 2151.585 | 1045.085 | 0.010757 | 0.010623 | 1763.379 |
| 180000 | 2130.095 | 1332.139 | 0.011523 | 0.011707 | 1843.483 |
| 190000 | 1974.066 | 1392.464 | 0.01245 | 0.011443 | 2244.833 |
| 200000 | 2546.532 | 1801.165 | 0.012382 | 0.014791 | 2394.682 |

## HeapSort

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | HeapSort |  |  |
| N | Losowe | Malejące | Rosnące | Stałe | V |
| 50000 | 0.44 | 0.22 | 0.22 | 0.03 | 0.38 |
| 60000 | 0.48 | 0.30 | 0.28 | 0.03 | 0.46 |
| 70000 | 0.57 | 0.34 | 0.32 | 0.04 | 0.48 |
| 80000 | 0.64 | 0.43 | 0.35 | 0.05 | 0.67 |
| 90000 | 0.79 | 0.42 | 0.47 | 0.05 | 0.71 |
| 100000 | 0.78 | 0.47 | 0.63 | 0.05 | 0.52 |
| 110000 | 0.94 | 0.71 | 0.51 | 0.06 | 1.17 |
| 120000 | 1.07 | 0.87 | 0.57 | 0.07 | 1.04 |
| 130000 | 1.02 | 0.58 | 0.65 | 0.07 | 1.37 |
| 140000 | 1.26 | 0.67 | 0.72 | 0.08 | 1.01 |
| 150000 | 1.30 | 1.04 | 0.99 | 0.08 | 1.26 |
| 160000 | 1.38 | 0.86 | 0.92 | 0.08 | 4.15 |
| 170000 | 2.78 | 0.88 | 0.81 | 0.10 | 1.47 |
| 180000 | 1.52 | 2.09 | 0.86 | 0.14 | 1.63 |
| 190000 | 2.03 | 1.55 | 0.85 | 0.11 | 1.55 |
| 200000 | 2.08 | 1.00 | 0.92 | 0.12 | 2.08 |

# Zaimplementuj algorytm Quicksort w dwóch wersjach: rekurencyjnie oraz iteracyjnie (z własną implementacją  stosu).  Porównaj obie wersje na wspólnym wykresie przy sortowaniu ciągu losowego. Następnie porównaj różne  sposoby wyboru klucza do porównania: skrajnie prawego, środkowego co do położenia, losowo wybranego.  Utwórz wykres porównujący efektywność działania algorytmu (iteracyjnego lub rekurencyjnego) w zależności od  wyboru różnego klucza dla A‐kształtnego ciągu wejściowego (przynajmniej 15 punktów pomiarowych).

# QuickSort rekurencyjny / iteracyjny

|  |  |  |
| --- | --- | --- |
|  | Quicksort |  |
| N | Rekurencyjny | Iteracyjny |
| 50000 | 0.54 | 0.57 |
| 60000 | 0.49 | 0.53 |
| 70000 | 0.58 | 0.75 |
| 80000 | 0.71 | 0.80 |
| 90000 | 1.02 | 0.96 |
| 100000 | 0.95 | 0.96 |
| 110000 | 1.17 | 1.05 |
| 120000 | 1.25 | 1.03 |
| 130000 | 1.39 | 1.17 |
| 140000 | 1.42 | 1.35 |
| 150000 | 1.41 | 1.45 |
| 160000 | 1.60 | 1.54 |
| 170000 | 1.77 | 1.54 |
| 180000 | 1.78 | 2.47 |
| 190000 | 1.89 | 2.03 |
| 200000 | 2.16 | 2.06 |

## Sposób wyboru klucza dla algorytmu QuickSort

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R - wersja rekurencyjna, I - wersja iteracyjna | | | | |  |  |  |  |
| N | R - lewy | I - lewy | R - prawy | I - prawy | R - środkowy | I - środkowy | R - losowy | I - losowy |
| 50000 | 0.36 | 0.45 | 0.38 | 0.37 | 0.33 | 0.34 | 0.95 | 1.00 |
| 60000 | 0.44 | 0.45 | 0.45 | 0.39 | 0.44 | 0.45 | 0.54 | 0.93 |
| 70000 | 0.66 | 0.56 | 0.47 | 0.45 | 1.39 | 0.59 | 0.60 | 0.71 |
| 80000 | 0.53 | 0.57 | 0.57 | 0.54 | 0.57 | 0.58 | 0.70 | 0.72 |
| 90000 | 0.73 | 0.72 | 0.66 | 0.61 | 0.64 | 0.74 | 0.86 | 0.83 |
| 100000 | 0.88 | 0.81 | 0.75 | 0.86 | 0.72 | 0.85 | 1.05 | 0.94 |
| 110000 | 0.85 | 1.20 | 0.90 | 0.81 | 0.75 | 0.84 | 1.02 | 1.06 |
| 120000 | 1.28 | 0.98 | 0.99 | 1.41 | 0.94 | 0.99 | 1.33 | 1.04 |
| 130000 | 0.99 | 1.07 | 0.99 | 0.90 | 0.94 | 1.00 | 1.17 | 1.37 |
| 140000 | 1.21 | 1.37 | 1.06 | 1.11 | 1.42 | 1.46 | 1.36 | 1.32 |
| 150000 | 1.15 | 1.59 | 1.38 | 1.23 | 1.59 | 1.56 | 1.30 | 1.51 |
| 160000 | 1.39 | 1.28 | 1.17 | 1.29 | 1.10 | 1.11 | 1.63 | 1.55 |
| 170000 | 1.23 | 1.77 | 2.29 | 2.50 | 1.66 | 1.48 | 1.63 | 1.69 |
| 180000 | 1.52 | 1.97 | 1.42 | 1.39 | 1.65 | 1.29 | 1.84 | 2.66 |
| 190000 | 1.53 | 1.53 | 1.82 | 1.66 | 1.50 | 1.69 | 1.99 | 1.88 |
| 200000 | 1.57 | 1.55 | 1.72 | 1.64 | 2.78 | 1.59 | 2.10 | 2.04 |

# Podsumowanie

Algorytm **InsertionSort** osiągnął średnią złożoność obliczeniową O(n^2). W przypadku danych rosnących lub stałych (wariant optymistyczny) złożoność ta jest liniowa.

Algorytm **SelectionSort** osiągnął złożoność O(n^2) w każdym przypadku.

Algorytm **CoctailSort** osiągnął złożoność O(n^2), ale jeśli dane są rosnące to ta złożoność jest liniowa.

Algorytm **HeapSort** osiągnął średnią złożoność O(n\*log(n)).

Algorytm **QuickSort** osiągnął złożoność O(n\*log(n)), jednak pesymistyczna złożonośc jest kwadratowa O(n^2). Wersja rekurencyjna i iteracyjna nie, zgodnie z przypuszczeniami nie różnią się wynikami.