Politechnika Łódzka

Szybkie algorytmy

Techniki zwiększenie efektywności algorytmów

Prowadzący zajęcia:

prof. dr hab. Mykhaylo Yatsymirskyy

Autor:

Filip Rynkiewicz

2016-11-14

1 Algorytm sortowania przez wstawianie

Dla dwóch elementów

Początkowym krokiem algorytmu jest posortowanie par (x, y) w zbiorze V, gdzie każda para $(x, y) \in V$, tak aby pierwsza liczba x była zawsze liczba mniejsza od liczby y. Indeksy liczby x jest zawsze o jeden mniejszy od indeksu liczby y w zbiorze.
Pierwszym krokiem tego sortowania będzie wybranie pary (x', y'). Pary wybierane sa poprzez przesuwanie od indeksu 2 zbioru V, poniewaz zakladamy ze pierwsza para jest posortowana, zawsze o 2 indeksy. Kazde przejscie zaczyna sie od elementu $V[i + 2]$[1].
Po wybraniu pary (x', y') nastepuje porownywanie elementu y' z elementem z, ktory poprzedza wybrana pare oraz indeks $z \in |V|$. Dopoki $z > y'$ wykonuje sie przesuniecie calej pary przed liczbe z. Za kazdym razem liczba z jest liczba poprzedzajaca liczbe x'. Jezeli $z < y'$ algorytm przechodzi do porownania $z > x'$. Jezeli zostanie spelniony ten warunek liczba mniejsza z pary zostaje przestawiona przed liczba z.

Wyniki

[1]Indeks $i + 2$ ze zbioru V, gdzie i jest kolejna iteracja algorytmu.

Kod

```
void sort(std::vector<int> &toSort)
{
 const int sizeOfArray=toSort.size()-(toSort.size()%2);
 for(int i=0; i<sizeOfArray; i+=2)
     {
      if(toSort[i] > toSort[i+1])
         {
          std::swap(toSort[i],toSort[i+1]);
         }
     }
 for(int i=2; i<sizeOfArray; i+=2)
    {
      const int pom1 = toSort[i];
      const int pom2 = toSort[i+1];
      int j = i-1;
          while(j>=0 && toSort[j]>pom2)
           {
            toSort[j+2] =  toSort[j];
            j--;
           }
          toSort[j+2] = pom2;
          toSort[j+1] = pom1;
          while(j>=0 && toSort[j]>pom1)
           {
            toSort[j+1] = toSort[j];
            --j;
           }
          toSort[j+1] = pom1;
        }
        if(toSort.size()%2==1)
           {
            const int pom = toSort[toSort.size()-1];
            int k = toSort.size()-2;
            while(k>=0 && toSort[k]>pom)
             {
              toSort[k+1] = toSort[k];
              --k;
                 }
           toSort[k+1] = pom;
            }
}
```

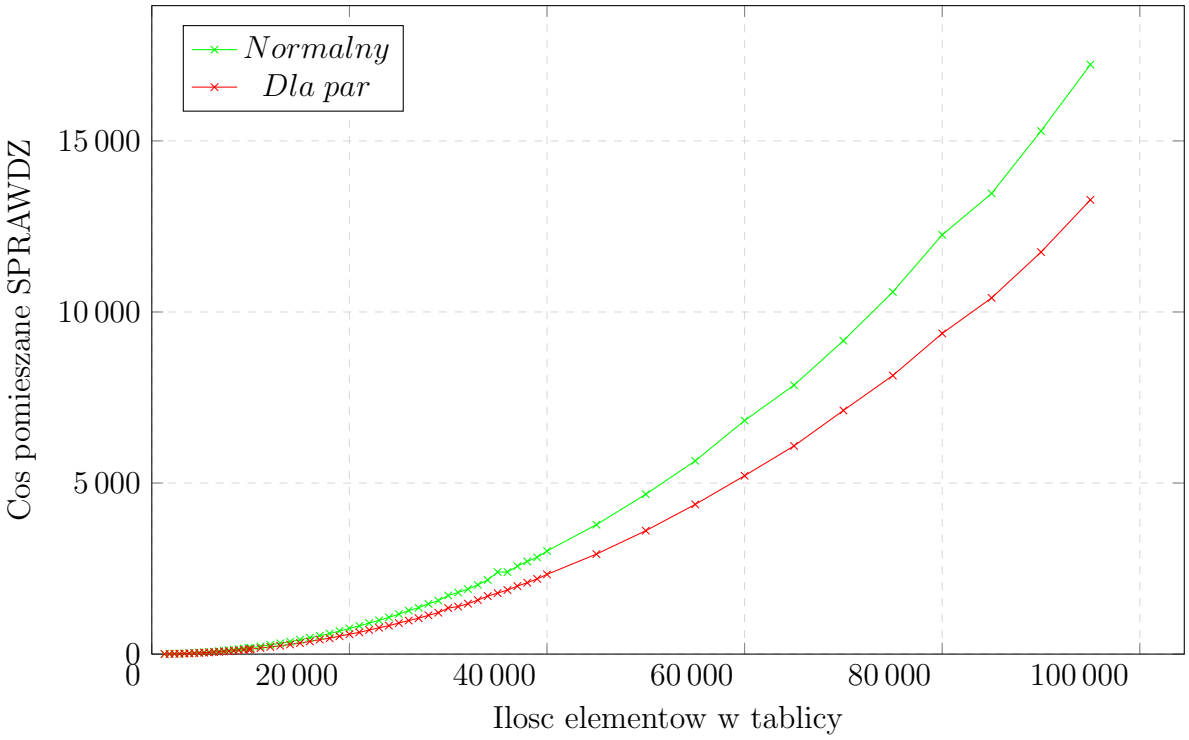

Figure 1: Wykres dla losowych elementow tablicy

2 Algorytm sortowania bąbelkowego

2.1 Dla dwóch elementów

Kod

```
void sort(std::vector<int> &toSort)
{
 for(int i= 0; i<(toSort.size()-1); i++)
 {
  int minElem=i,maxElem=i+1;
  if(toSort[minElem]>toSort[maxElem])
   {
    std::swap(toSort[minElem],toSort[maxElem]);
   }
  int j=i;
  while(j>0 && toSort[minElem]<toSort[minElem-1])
```

```
    {
     std::swap(toSort[minElem],toSort[minElem-1]);
     --j;
     minElem--;
    }
   int j2=i+1;
   while(j2<(toSort.size()-1) && toSort[maxElem]>toSort[
      maxElem+1])
    {
     std::swap(toSort[maxElem],toSort[maxElem+1]);
     ++j2;
     maxElem++;
    }
  }
}
```

Wyniki

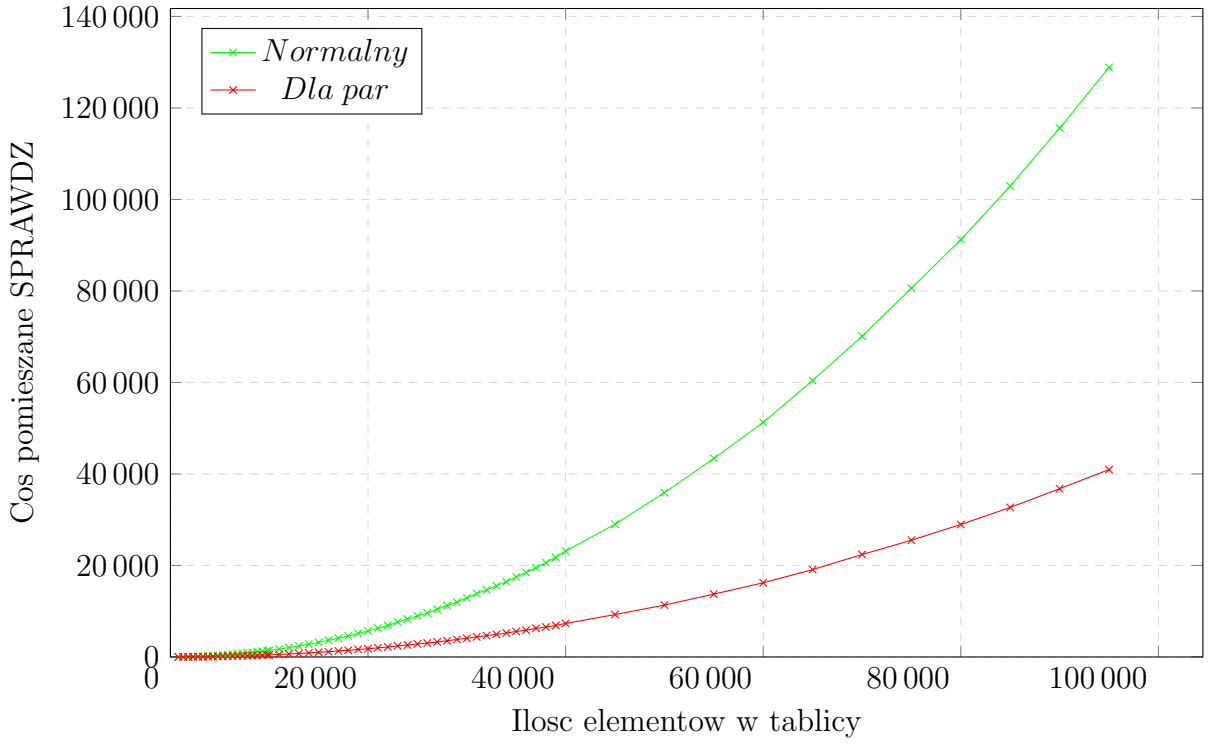

Figure 2: Wykres dla losowych elementow tablicy

3 Algorytm sortowania przez wybieranie

3.1 Dla dwóch elementów

Kod

```
void sort(std::vector<int> &toSort)
{
 int vectorSize=0;
 if(toSort.size()%2!=0)
  {
   vectorSize++;
   std::iter_swap((std::min_element(toSort.begin(),toSort.end
       ())),toSort.begin());
  }
 std::vector<int>::iterator _begin = toSort.begin()+
    vectorSize;
 std::vector<int>::iterator _end = toSort.end() - 1;
 while (_begin < _end)
  {
   std::vector<int>::iterator it=_begin,_min=it,_max=it;
   for (it = _begin; it <= _end; ++it)
    {
     if ((*it) < (*_min))
      {
           _min = it;
         }
        else if ((*it) > (*_max))
         {
          _max = it;
         }
    }
 std::iter_swap(_min,_begin);
 if(_begin==_max)
  {
   _max=_min;
  }
 std::iter_swap(_max,_end);
 ++_begin;
 --_end;
  }
}
```

Wyniki

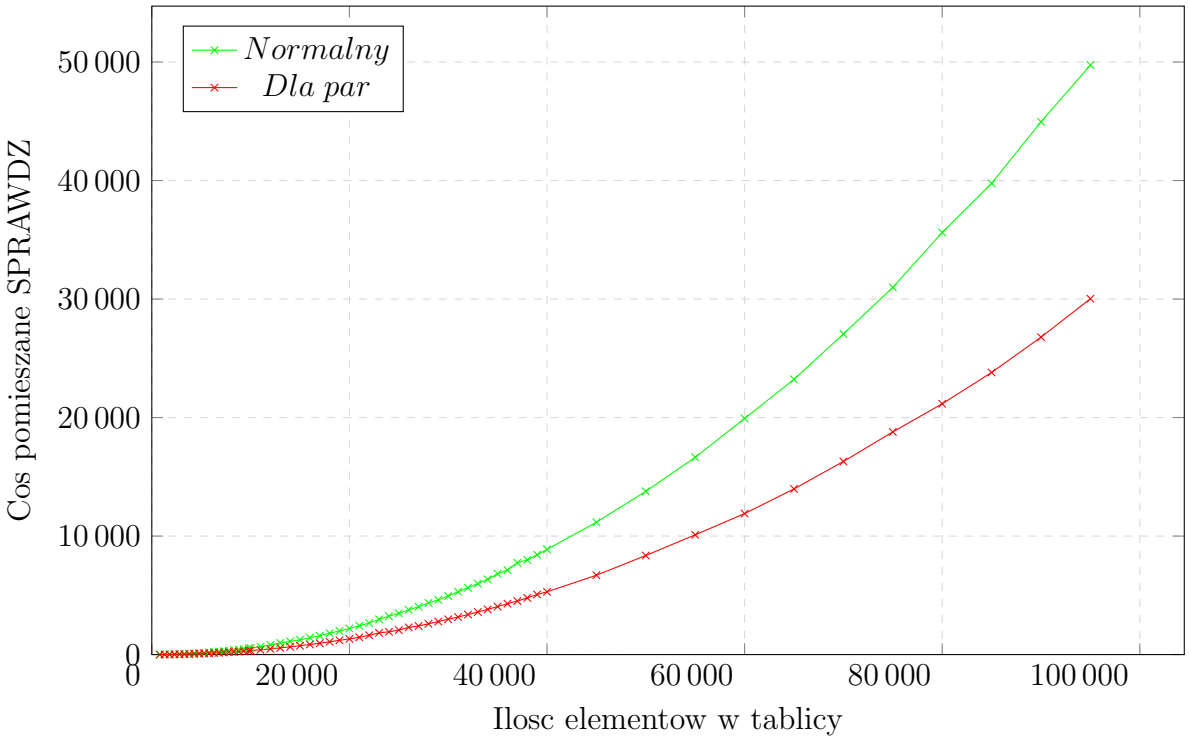

Figure 3: Wykres dla losowych elementow tablicy