Politechnika Wrocławska

Mikroprojekt 1

Projektowanie i analiza algorytmów

Wydział Informatyki i Telekomunikacji

Informatyczne Systemy Automatyki

Aleksander Żołnowski, 272536

1. Wstęp

Zadanie polegało na zaprojektowaniu i zaimplementowaniu algorytmu do zarządzania i sortowania pakietów. Każdy pakiet otrzymywał unikalny numer i miał być posortowany oraz złożony w prawidłowej kolejności. Celem było skonstruowanie efektywnego rozwiązania radzącego sobie z losowym odbiorem pakietów.

2. Wybór struktury danych

Jako strukturę danych przechowującą pakiety wykorzystano tablicę dynamiczną przechowującą struktury pakietów Packet<T>, gdzie każdy pakiet składa się z danych(T data) oraz numeru pakietu(int number).

```
template <class T>
struct Packet{
    T data;
    int number;
};
```

Rysunek 1 Implementacja struktury pakietu

Wybór tablicy dynamicznej był podyktowany jej zdolnością do efektywnego rozszerzania się przez dodawanie kolejnych pakietów, szybki dostęp do elementów oraz umożliwieniem implementacji algorytmu sortowania przez scalanie (merge sort).

```
template <class T>
class Array{
private:
    Packet<T>* array;
    int capacity;
    int size;
```

Rysunek 2 Fragment implementacja struktury danych - klasy tablicy dynamicznej

Klasa Array jest odpowiedzialna za zarządzanie dynamiczną tablicą pakietów. Zawiera ona szereg metod obsługujących strukturę danych.

3. Proponowane rozwiązanie

Proponowanym rozwiązaniem problemu jest implementacja algorytmu sortującego, sortowanie przez scalenie (ang. merge sort). Jest to algorytm działający według zasady „dziel i zwyciężaj”. Działa on w następujący sposób:

1. Rekurencyjnie dzieli tablicę na dwie połowy
2. Sortuje każdą z połówek
3. Scalanie – łączy ze sobą posortowane połówki w posortowaną tablice.

```
template <class T>
void Array<T>::mergeSort(Packet<T>* arr, int left, int right)
{
    if (left >= right) return;

    int mid = left + (right - left) / 2;
    mergeSort(array, left, mid);
    mergeSort(array, mid + 1, right);
    merge(array, left, mid, right);
}
```

Rysunek 3 Implementacja metody mergeSort

```
template <class T>
void Array<T>::merge(Packet<T>* arr, int left, int mid, int right)
{
    int size1 = mid - left + 1;
    int size2 = right - mid;

    Packet<T>* leftArray = new Packet<T>[size1];
    Packet<T>* rightArray = new Packet<T>[size2];

    for (int i = 0; i < size1; i++) {
        leftArray[i] = arr[left + i];
    }

    for (int j = 0; j < size2; j++) {
        rightArray[j] = arr[mid + 1 + j];
    }
    int i = 0;
    int j = 0;
    int k = left;

    while (i < size1 && j < size2) {
        if (leftArray[i].number <= rightArray[j].number)
        {
            arr[k] = leftArray[i];
            i++;
        }
        else {
            arr[k] = rightArray[j];
            j++;
        }
        k++;
    }

    while (i < size1) {
        arr[k] = leftArray[i];
        i++;
        k++;
    }

    while (j < size2){
        arr[k] = rightArray[j];
        j++;
        k++;
    }

    delete[] leftArray;
    delete[] rightArray;
}
```

Rysunek 4 Implementacja metody merge

Sortowanie odbywa się dla numerów pakietów w strukturze Packet<T>. Każdy z pakietów ma unikalny numer, co zapewnia prawidłowe posortowanie tablicy pakietów.

Kolejnym krokiem jest złożenie posortowanych pakietów w jedną wiadomość.

```
template <class T>
string Array<T>::getMessage()
{
    ostringstream convert;
    if(isEmpty()==true)
    {
        return "";
    }
    else
    {
        for(int i=0; i<size; i++)
        {
            convert << array[i];
        }
        string message = convert.str();
        return message;
    }
}
```

Rysunek 5 Implementacja metody getMessage

Powyżej przedstawiona metoda odpowiada za złożenie wszystkich pakietów przechowywanych w tablicy w jedną całą wiadomość typu string.

4. Złożoność obliczeniowa

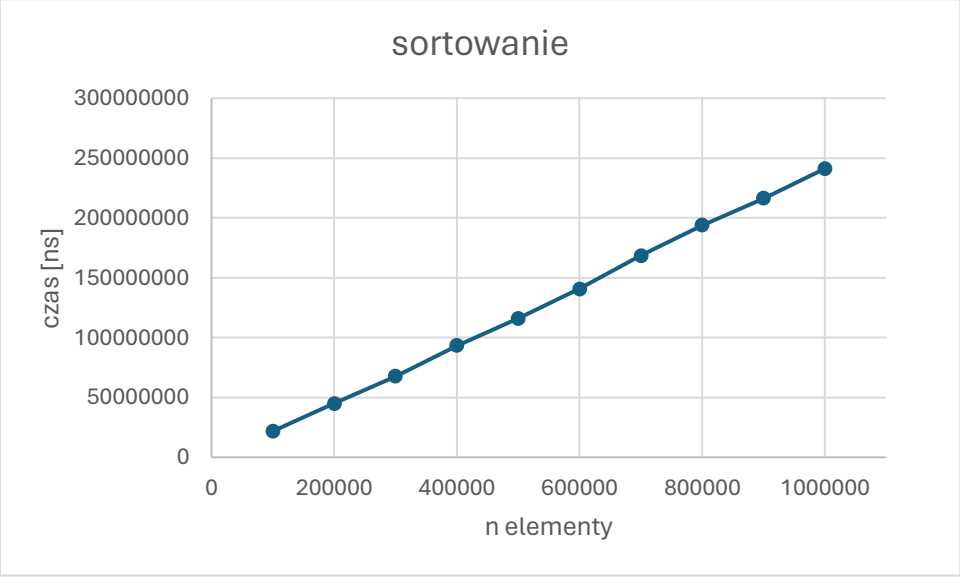
4.1 Algorytm sortowania

W celu wyznaczenia wydajności algorytmu merge sort, przeprowadzono serię testów operacji sortowania. Badania zostały przeprowadzone na różnych ilościach n pakietów (100000, 200000, …, 10e6). Wydajność została zmierzona w nanosekundach(ns).

sortowanie	
n	t[ns]
100000	21800625
200000	44988291
300000	67714750
400000	93382958
500000	116127500
600000	140655667
700000	168564583
800000	193867125
900000	216410750
1000000	241125500

Tabela 1 Wyniki sortowania

Algorytm merge sort dzieli rekurencyjnie tablicę na pół, do czasu aż każda z części będzie miała długość 1. Następnie scala je ze sobą. Każde takie scalenie to koszt O(n). W związku z tym, że tablicę wejściową dzieliliśmy za każdym razem na pół, takich scaleń mamy log(n). Złożoność obliczeniowa wynosi zatem O(nlog(n)).



Rysunek 6 Charakterystyka złożoności obliczeniowej sortowania

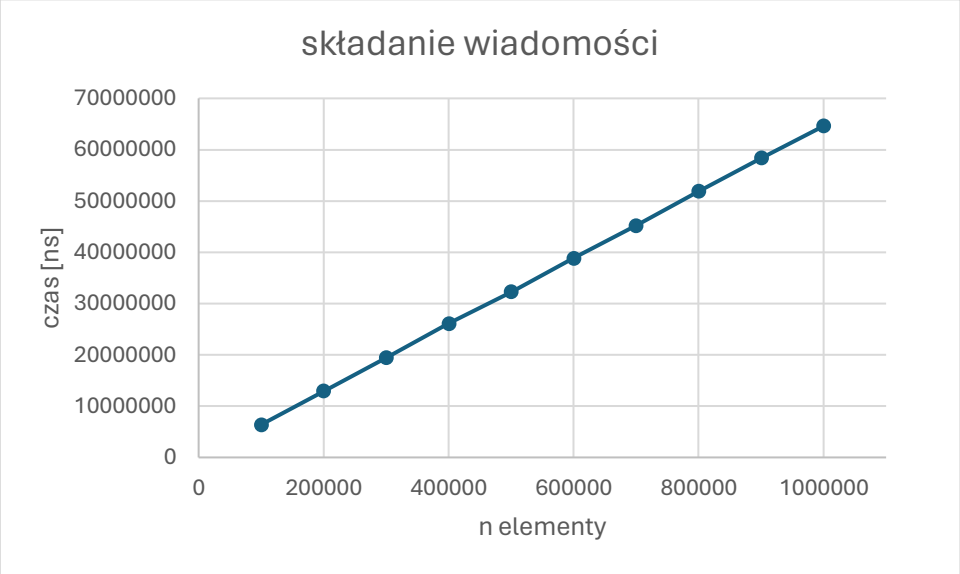
4.2 Składanie wiadomości

Analogicznie zbadano wydajność metody składającej wiadomość dla różnych ilości n pakietów. Wydajność została również zmierzona w nanosekundach (ns).

składanie wiadomości	
n	t[ns]
100000	6386083
200000	12939791
300000	19395334
400000	26114917
500000	32275916
600000	38796000
700000	45176792
800000	51881458
900000	58368791
1000000	64605375

Tabela 2 Wyniki składania wiadomości

Metoda iteruje po całej tablicy i konwertuje dane pakietów w celu złożenia wiadomości. Koszt składania wiadomości jest O(n). Widać, że złożoność czasowa rośnie liniowo zgodnie z n, co potwierdza liniową złożoność.



Rysunek 7 Charakterystyka złożoności obliczeniowej składania wiadomości

5. Wnioski

• Efektywność sortowania przez scalanie:
Z przeprowadzonych badań wynika, że algorytm sortowania przez scalanie (merge sort) jest bardzo efektywny dla dużej ilości pakietów. Jego złożoność obliczeniowa wynosi O(nlog(n)), co potwierdzają uzyskane wyniki.

• Stabilność i dokładność algorytmu:
Algorytm sortowania przez scalanie zachowuje stabilność, co jest kluczowe w kontekście prawidłowego sortowania dużych ilości pakietów.

• Liniowa złożoność składania wiadomości:
Metoda składania wiadomości charakteryzuje się liniową złożonością czasową O(n). Oznacza to, że czas potrzebny do złożenia wiadomości rośnie w sposób liniowy wraz ze wzrostem liczby pakietów.

6. Bibliografia

- Data Structures and Algorithms in C++, 2nd Edition Michael T. Goodrich, Roberto Tamassia, David M. Mount
- Cormen T., Leiserson C.E., Rivest R.L., Stein C., Wprowadzenie do algorytmów, WNT
- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie