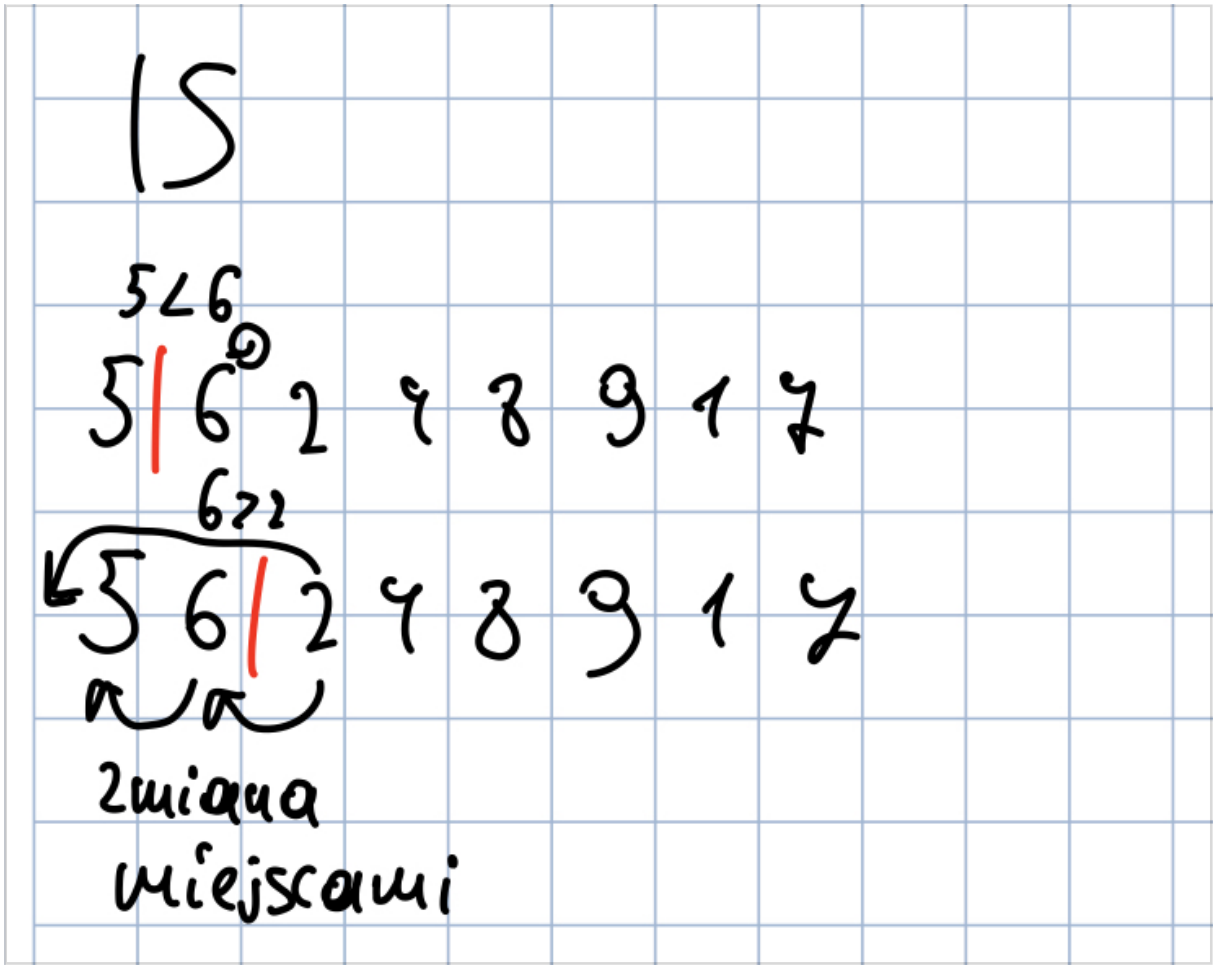
LAB 1 - SORT

Proste - O(n^2)

IS - proste wstawianie

Tablica na dwie części i porównujemy ich wartości (zmiana miejscami)




```
for (int i = 1; i < n; i++) {
        for (int j = i; j > 0; j--) {
            if (arr[j] < arr[j-1]) {
                swap(arr[j],arr[j-1]);
            }
        }
    }
```

Efektywność pamięciowa - działa na miejscu
Złożoność czasowa: O(n^2) - maksymalna liczba kroków wykonanych na DTM(Determined Turing machine) dla instancji o określonym rozmiarze.

Przypadek najlepszy - posortowana tablica O(n), to jest zaleta, bo przyspiesza dla elementów posortowanych.

Przypadek najgorszy - posortowana odwrotnie tablica, O(n^2)

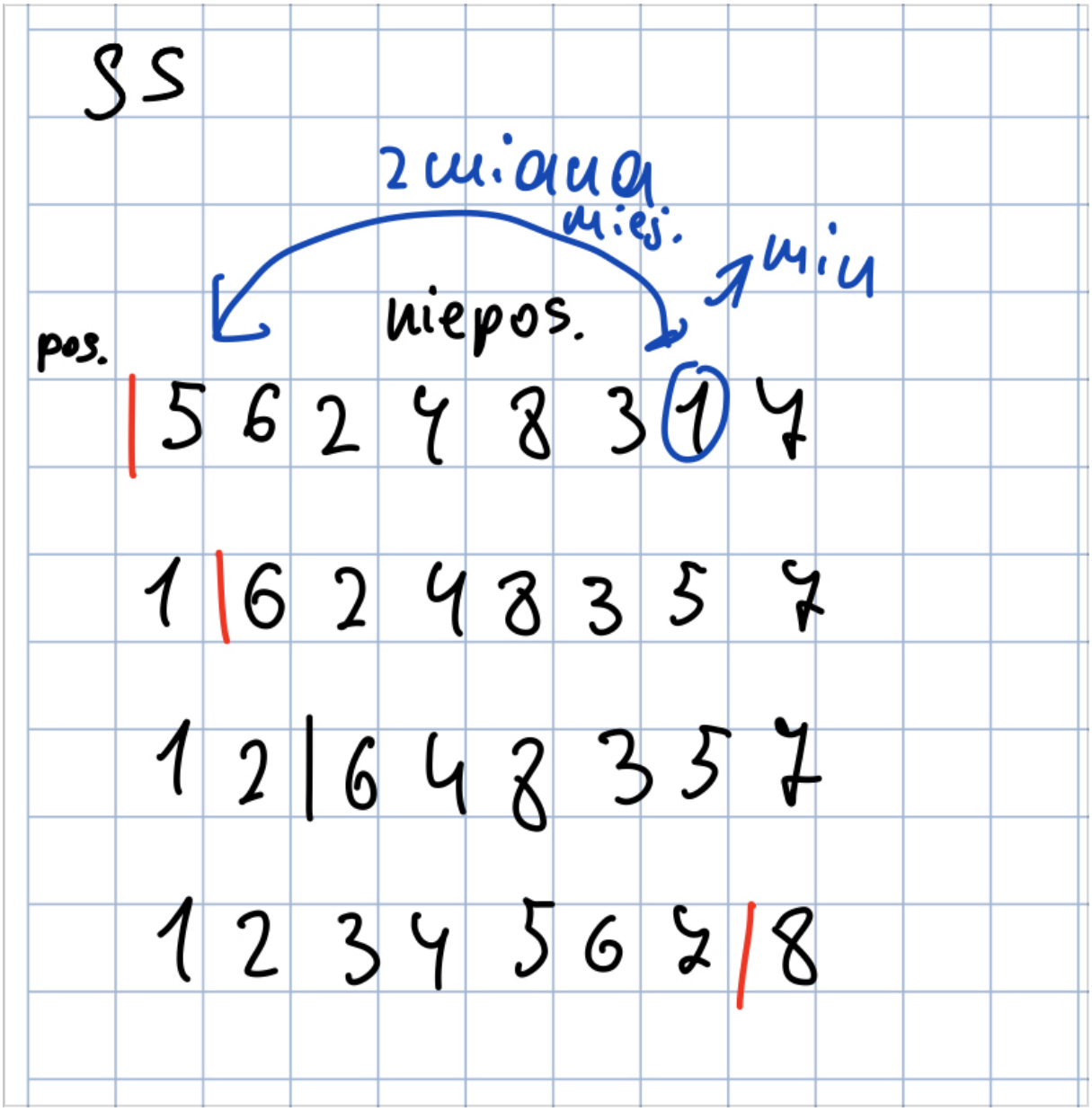
Ma zachowanie naturalne. - Jeśli dla danych wstępnie posortowanych (choćby częściowo) algorytm wykonuje się szybciej, niż dla zupełnie wymieszanych, to wówczas mówimy, że algorytm wykazuje zachowanie naturalne.

Wrażliwość na dane: jest wrażliwy na dane. (Bo dla 10 i dla 100 elementów różnica jest bardzo duża)

Przypadek średni: O(n^2)

Jest stabilny - zachowuje kolejność wejściową. 3 3 3

SS - selection sort




```
for (int i = 0; i < n - 1; i++) {
        int min = arr[i];
        int min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < min) {
                min = arr[j];
                min_idx = j;
            }
        }
        swap(arr[i], arr[min_idx]);
    }
```

Przypadek najlepszy = **Przypadek najgorszy** = **Przypadek średni**: O(n^2)
Nie jest wrażliwy na układ danych
Bardzo słabe zachowanie naturalne
Działa w miejscu
nie jest stabilny

BS - bąbelkowe

```
for (int i = 0; i < n; i++){
        for (int j = n - 1; j > i; j--) {
            if (arr[j] < arr[j-1]) {
                swap(arr[j], arr[j-1]);
            }
        }
    }
```

Złożoność - O(n^2) we wszystkich przypadkach
Działa w miejscu
Jest stabilny
Nie jest wrażliwy na układ danych
Ulepszenie:
1) Żeby ułatwić, można postawić flagę czy była zmiana, czy nie(bo kiedy dwie części są posortowane, nadal będą wykonywać się operacje zmiany), wynik ➜ przypadek najlepszy O(n).
2) Zapamiętanie miejsca zmiany.

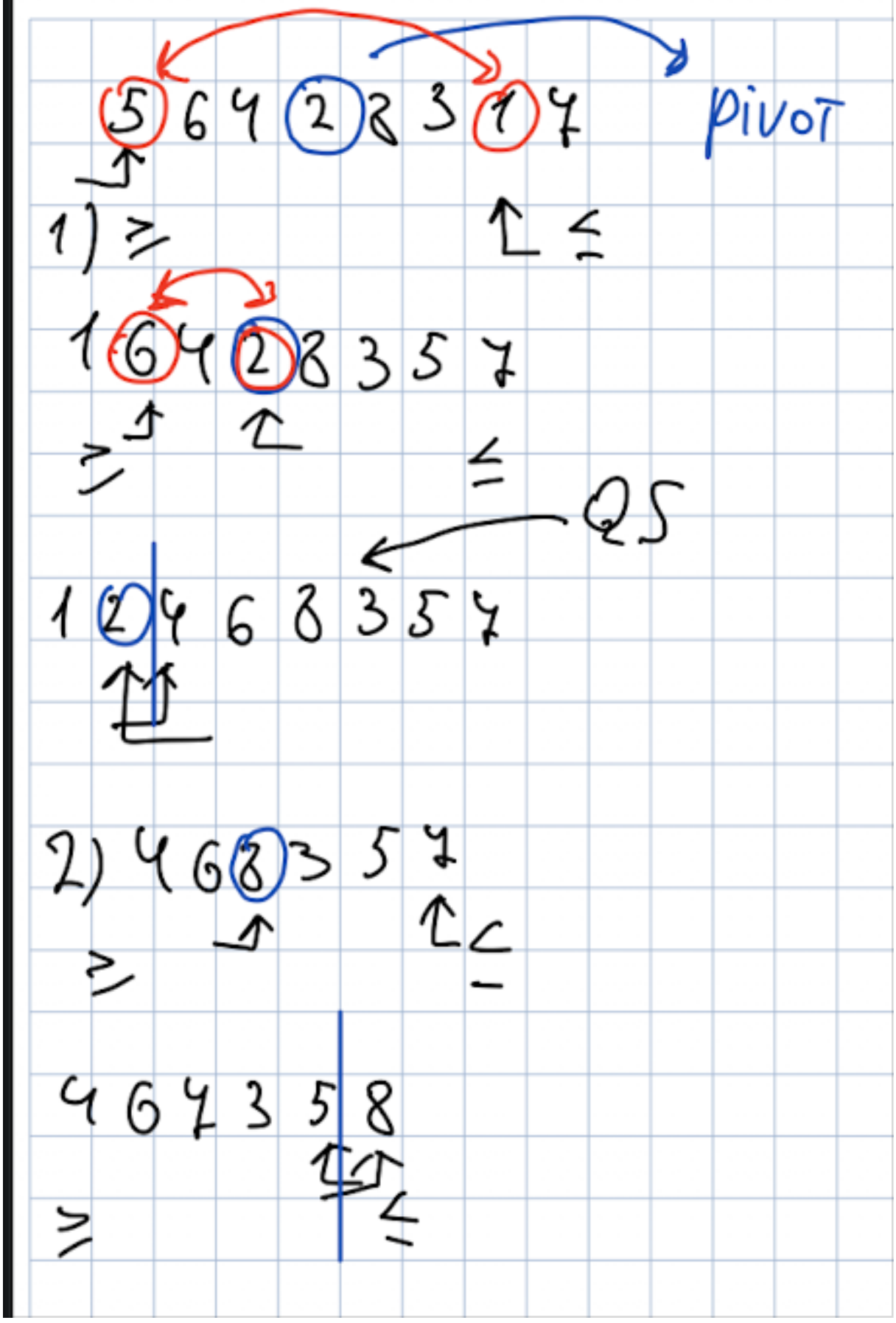
MS - metoda shella
Sortowanie przez scalenie + proste wstawianie, **Przypadek średni** - O(n^1.2)

```
for (int d = n / 2; d > 0; d /= 2) {
        for (int i = d; i < n; i++) {
            for(int j = i - d; j >= 0; j -= d) {
```

```
                if (arr[j+d] < arr[j]) {
                    swap(arr[j+d], arr[j]);
                }
            }
        }
    }
```

QS - quick sort

Przypadek pesymistyczny (O(n^2)) - element o skrajnej wartości (najmniejszy, największy)
Przypadek optymistyczny (logn) - kiedy pivot jest medianą
Przypadek średni (nlogn)
Nie zachowuje się naturalnie
Jest bardzo wrażliwy na dane
Nie jest stabilny
Złożoność pamięciowa - działanie na miejscu, ale rekurencja potrzebuje pamięć
Ma zachowanie naturalne

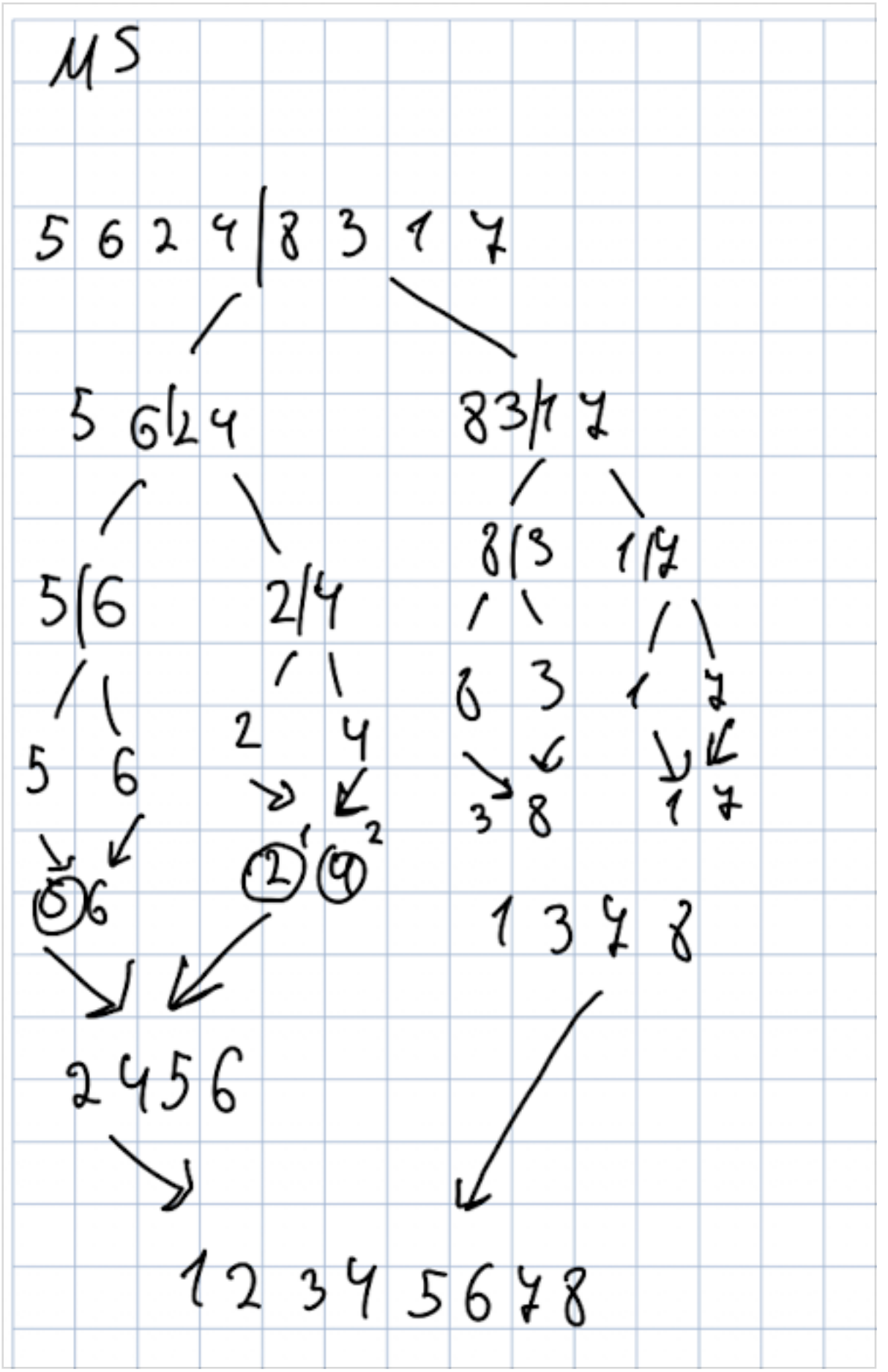

5 6 4 2 8 3 1 7
pivot
1) ≥
≤
1 6 4 2 8 3 5 7
≥
≤
25
1 2 4 6 8 3 5 7
2) 4 6 8 3 5 7
≥
≤
4 6 4 3 5 8
≥
≤

```
void QS(vector<int> &arr,int left, int right) {

    if (left >= right) {
        return;
    }

    int l = left;
    int r = right;
    int piv_idx = (left + right) / 2;
    double pivot = arr[piv_idx];
    swap(arr[piv_idx], arr[right]);
    piv_idx = right;
    right--;

    while (right >= left) {

       if (arr[left] >= pivot) {
            if (arr[right] <= pivot) {
                swap(arr[left],arr[right]);
                left++;
                right--;
            }
            else {
                right--;
            }
        }
        else if (arr[right] <= pivot) {
            left++;
        }
        else {
            left++;
            right--;
        }
    }

    swap(arr[left],arr[piv_idx]);
    QS(arr,l,right);
    QS(arr,left+1,r);
```

```
}
```

Sortowanie przez scalanie - MS

Przypadek najgorszy=Przypadek średni=Przypadek najlepszy O(nlogn)*
Nie działa w miejscu (bo wynik przepisujemy w nową tablicę) - O(n)
Jest stabilny
Nie jest wrażliwy na dane


MS
5 6 2 4 | 8 3 1 7
5 6 | 2 4
8 3 | 1 7
5 | 6
2 | 4
8 | 3
1 | 7
5 6 2 4 8 3 1 7
2 4 5 6
1 3 7 8
1 2 3 4 5 6 7 8

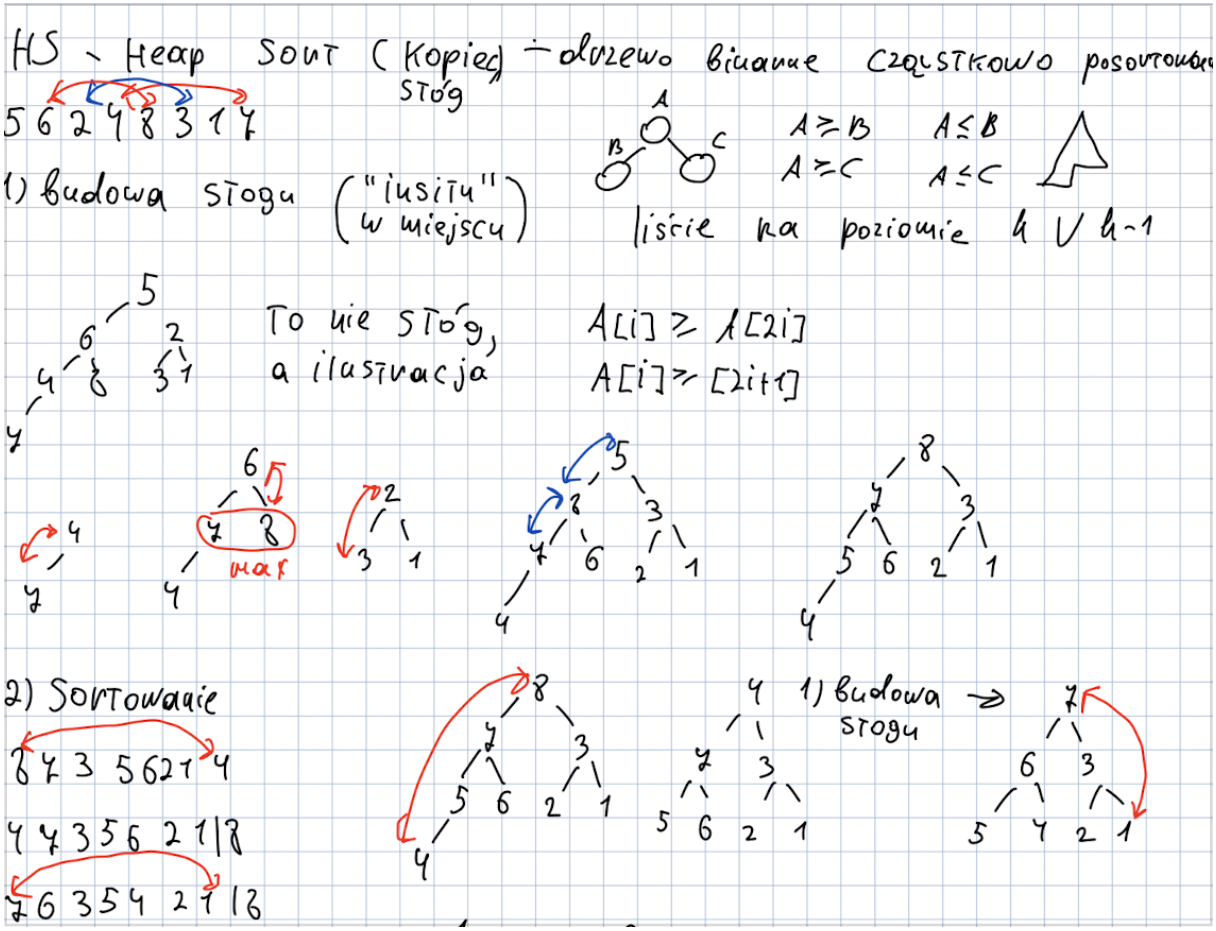
```
vector<int> MS(vector<int> arr, int n) {

    if (n == 2) {
        if (arr[0] > arr[1]) {
            swap(arr[0],arr[1]);
        }
    }

    if (n == 1) {
        return arr;
    }

    vector<int> arr_l;
    vector<int> arr_r;

    arr_l = vector<int>(arr.begin(), arr.begin()+((n-1)/2)+1);
    arr_r = vector<int>(arr.begin()+((n-1)/2)+1,arr.end());

    vector<int> left = MS(arr_l, arr_l.size());
    vector<int> right = MS(arr_r, arr_r.size());
    vector<int> merge;

    int left_size = left.size();
    int right_size = right.size();

    while (left_size > 0 && right_size> 0) {
        if (left[0] <= right[0]) {
            merge.push_back(left[0]);
            left.erase(left.begin());
            left_size--;
        }
        else {
            merge.push_back(right[0]);
            right.erase(right.begin());
            right_size--;
        }
    }
    if (left_size < 1) {
        merge.insert(merge.end(),right.begin(),right.end());
```

```
    }
    else if (right_size < 1) {
        merge.insert(merge.end(),left.begin(),left.end());
    }

    return merge;
}
```

HS - sortowanie stogowe

Budowa stogu:
Przypadek najgorszy O(logn)
Przypadek najlepszy O(1)
Sortowanie stogu:
Przypadek najgorszy=Przypadek średniO(nlogn)
Przypadek najlepszy O(n)

Jest wrażliwy na dane
Nie jest stabilny
Działa w miejscu

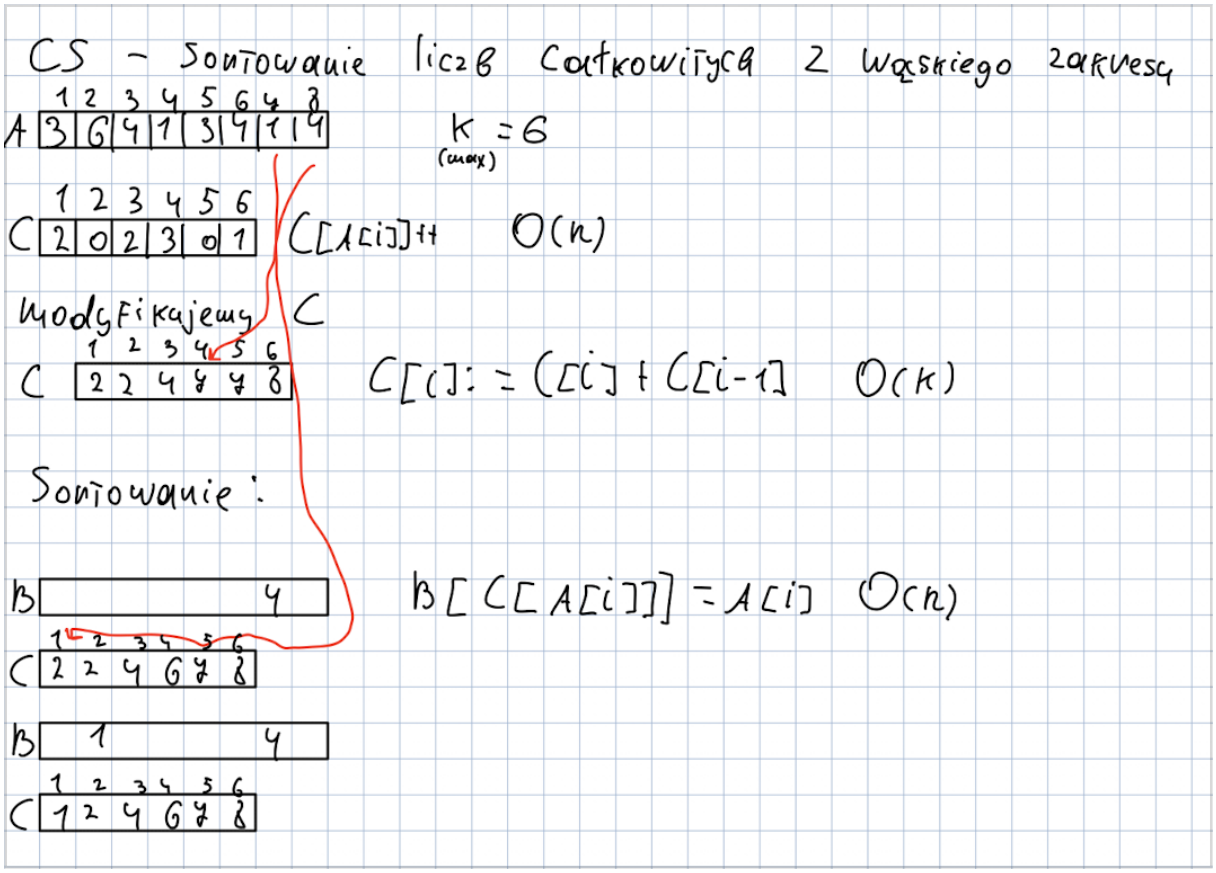



```
void Heapify(vector<int> &arr, int r, int n) {
    int largest = r;
    int left = r * 2 + 1;
    int right = r * 2 + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }

    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest != r) {
        swap(arr[r], arr[largest]);
        Heapify(arr, largest, n);
    }
}
```

```
void BuildHeap(vector<int> &arr, int n) {

    for (int i = n / 2 - 1; i >= 0; i--) {
        Heapify(arr,i, n);
    }
}

void HS(vector<int> &arr, int n) {

    BuildHeap(arr,n);
    for (int i = n-1; i > 0; i--) {
        swap(arr[0], arr[i]);
        Heapify(arr,0,i);
    }
    return;
}
```

CS

Nie działa w miejscu O(n+k)
Przypadek najgorszy O(n+k)
Jest stabilny
Jest wrażliwy na dane




```
void CS(vector<int> &arr, int n) {

    int k = arr[0];

    for (int i = 0; i < n; i++) {
        if (arr[i] > k) {
            k = arr[i];
        }
    }
    vector<int> store(k+1);
    for (int i = 0; i < k+1; i++) {
        store[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        store[arr[i]]++;
    }

    for (int i = 1; i < k+1; i++) {

        store[i] += store[i-1];
```

```
    }
    vector<int> final(n);
    for (int i = 0; i < n; i++) {
        final[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        final[store[arr[i]]-1] = arr[i];
        store[arr[i]]--;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = final[i];
    }

}
```

#University/Semester2/Algorytmy