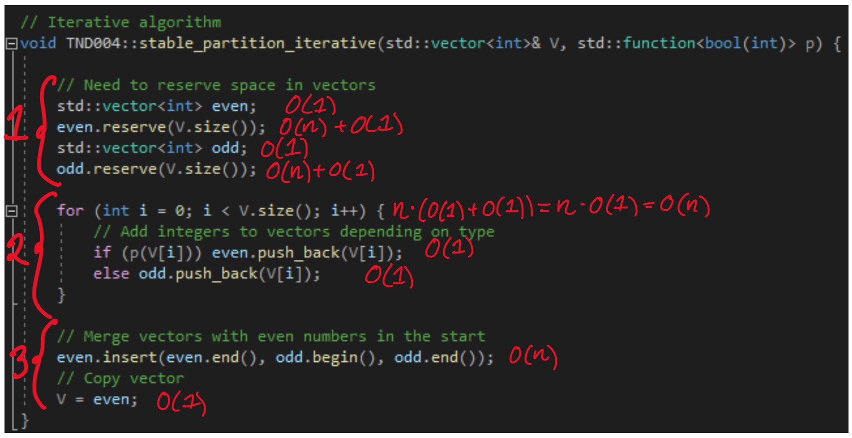
Lab1 analysis

Friday, April 8, 2022 12:12 PM



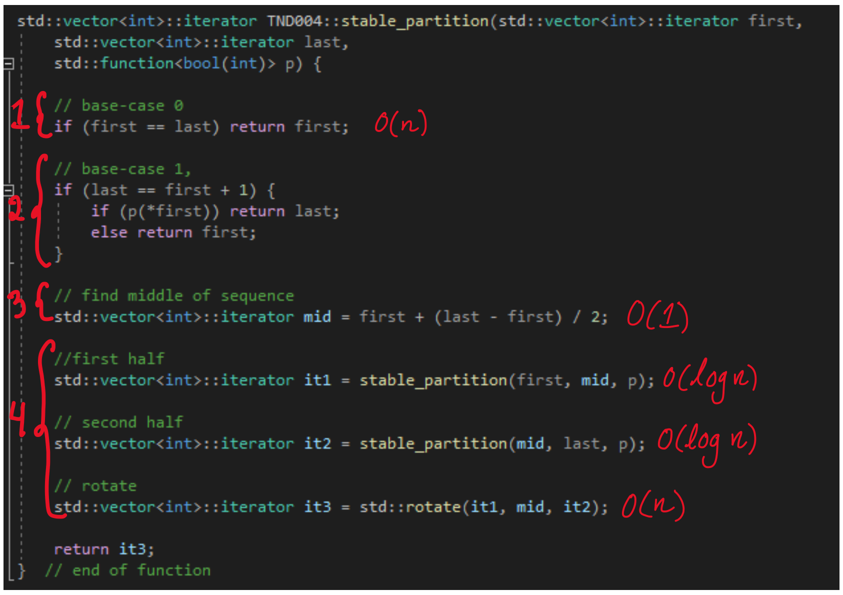
```
// Iterative algorithm
void TND004::stable_partition_iterative(std::vector<int>& V, std::function<bool(int)> p) {

    1 { // Need to reserve space in vectors
        std::vector<int> even;   O(1)
        even.reserve(V.size());  O(n) + O(1)
        std::vector<int> odd;  O(1)
        odd.reserve(V.size());  O(n)+O(1)

    2 { for (int i = 0; i < V.size(); i++) {  n·(O(1)+O(1)) = n·O(1) = O(n)
            // Add integers to vectors depending on type
            if (p(V[i])) even.push_back(V[i]);   O(1)
            else odd.push_back(V[i]);      O(1)
        }

    3 { // Merge vectors with even numbers in the start
        even.insert(even.end(), odd.begin(), odd.end());  O(n)
        // Copy vector
        V = even;  O(1)
}
```

Explanation

1. To initialize the vector does not depend on the amount of slots. We however need to reserve space to save complexity in future operations. The complexity of standard function *V.size()* is according to c++ reference always constant. The *reserve* function is depending on the size of vector and therefore is the complexity linear (confirmed by cppreference).
   Sum of part 1: O(1) + O(n) + O(1) + O(1) + O(n) + O(1) = **O(n)**
2. *Push_back* of a vector is not depending on size and therefor constant (according to cppreference it is amortized constant, but since we have reserved enough space in the vector it will stay constant). The complexity of the loop is *n times the complexity inside* of the loop (se equation).
   Sum of part 2: n * (O(1) + O(1)) = n * (O(1)) = **O(n)**
3. *Insert* is according to cpprefference linear in the distance between pos and end of the container. Assigning a variable is O(1)??? Or O(n??)
   Sum of part 3: O(n) + O(1) = **O(n)**

Sum of exercise 1:
O(n) + O(n) + O(n) = **O(n)**

Best case:
Worst case:



```
std::vector<int>::iterator TND004::stable_partition(std::vector<int>::iterator first,
    std::vector<int>::iterator last,
    std::function<bool(int)> p) {

    1 { // base-case 0
        if (first == last) return first;   O(n)

    2 { // base-case 1,
        if (last == first + 1) {
            if (p(*first)) return last;
            else return first;
        }

    3 { // find middle of sequence
        std::vector<int>::iterator mid = first + (last - first) / 2;  O(1)

    4 { //first half
        std::vector<int>::iterator it1 = stable_partition(first, mid, p);  O(log n)

        // second half
        std::vector<int>::iterator it2 = stable_partition(mid, last, p);  O(log n)

        // rotate
        std::vector<int>::iterator it3 = std::rotate(it1, mid, it2);  O(n)

    return it3;
}  // end of function
```

Explanation

1.
2.
3. Not depending on input size
   Sum of part 3: **O(1)**
4. According to cppreference
   Sum of part 4: O(log n) + O(log n) + O(n) = **O(n)**

Sum of exercise 2:

Best case:
Worst case: