ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«**[Split Array Largest Sum](https://leetcode.com/problems/split-array-largest-sum/)**»

Выполнил работу

Фамилия Имя : Торши Ромдхан

Академическая группа №группы: j3114

ИСУ: 467746

факультет цифровых трансформаций

Принято

Фамилия Имя преподавателя : Maxim Dunaev & Иван Ходненко

Санкт-Петербург

2024

1. **Введение:**  
   Цель задачи разделить массив ***nums*** на ***k*** непустых подмассивов таким образом, чтобы наибольшая сумма среди этих подмассивов была минимизирована. Для эффективного решения задачи используется двоичный поиск в сочетании с жадным методом.
2. **Теоретическая подготовка**

#### **Двоичный поиск:**

#### Двоичный поиск используется для нахождения минимальной наибольшей суммы, которая позволяет разделить массив на **k** подмассивов. Для этого необходимо рассмотреть диапазон возможных сумм: от максимального элемента массива (поскольку нельзя иметь сумму подмассива меньше максимального элемента) до суммы всех элементов массива (максимальная сумма, если будет только один подмассив).

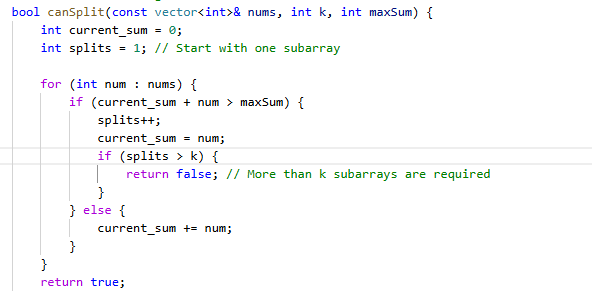
1. Жадная проверка (canSplit):

Для каждой возможной суммы (передаваемой в функцию) проверяется, можно ли разделить массив на ***k*** или меньше подмассивов, не превышающих эту сумму.

* + Если сумма текущего подмассива превышает максимально допустимую, то начинается новый подмассив.
  + Если количество подмассивов превышает ***k***, то возвращается ***false***.

1. **ДИЗАЙН АЛГОРИТМА**

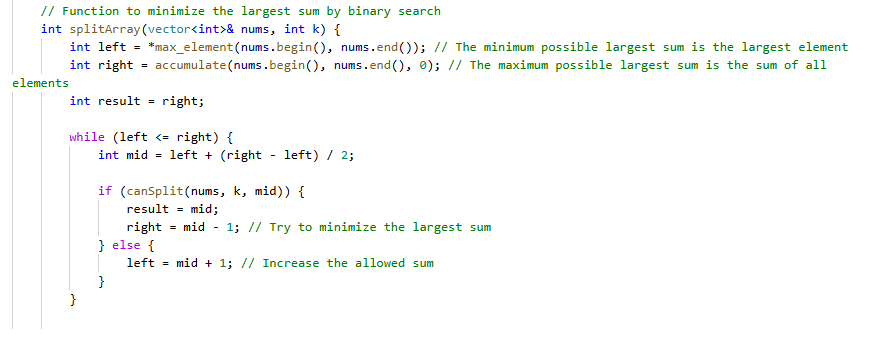
#### **3.1 Функция canSplit**



Эта функция проверяет, возможно ли разделить массив на k или меньше подмассивов, так чтобы наибольшая сумма не превышала заданное значение maxSum.

* Инициализируем current\_sum как 0 и splits как 1 (начинаем с первого подмассива).
* Проходим по массиву:
  + Добавляем текущий элемент к current\_sum. Если сумма превышает maxSum, создаем новый подмассив (увеличиваем splits и сбрасываем current\_sum в текущий элемент).
  + Если количество подмассивов превышает k, возвращаем false.

#### **3.2 Функция splitArray**



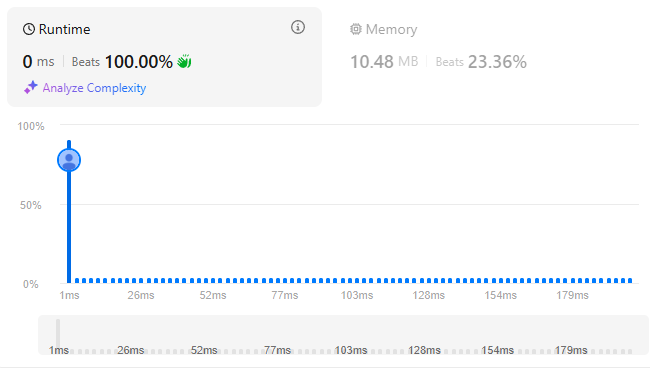
Эта функция использует двоичный поиск для нахождения минимальной наибольшей суммы, для которой возможно разделить массив.

* **Инициализация**:
  + Минимальная возможная наибольшая сумма — это максимальный элемент массива (left).
  + Максимальная возможная наибольшая сумма — это сумма всех элементов массива (right).
* **Цикл двоичного поиска**:
  + Средняя точка (mid) представляет кандидат для наибольшей суммы.
  + Используем функцию canSplit для проверки, возможно ли разделить массив при данной сумме.
  + Если canSplit возвращает true, это означает, что возможно уменьшить наибольшую сумму, поэтому сужаем правую границу.
  + Если canSplit возвращает false, текущая сумма слишком мала, увеличиваем левую границу.

1. **Экспериментальная часть**

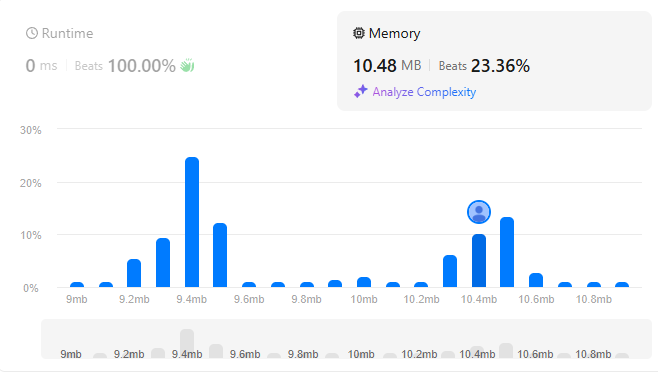
#### **Анализ сложности по времени**

* **Инициализация**:
  + Нахождение максимального элемента в массиве занимает **O(n)** времени **(left = \*max\_element(nums.begin(), nums.end())).**
  + Суммирование всех элементов массива занимает **O(n)** времени **(right = accumulate(nums.begin(), nums.end(), 0)).**
* **Цикл двоичного поиска**:
  + Каждый шаг двоичного поиска занимает **O(log(sum))**, где **sum** это сумма всех элементов массива.
  + На каждом шаге вызывается функция **canSplit**, которая проверяет, можно ли разделить массив с данной суммой, и она занимает **O(n)** времени.
  + Общая сложность по времени составляет **O(n log(sum))**.



#### **Анализ сложности по памяти**

* Пространственная сложность для дополнительных переменных и функции canSplit составляет **O(1)**.
* Пространственная сложность для массива данных — **O(n)**.

****

****

1. **Заключение**

Использование двоичного поиска в сочетании с жадным методом позволяет эффективно минимизировать наибольшую сумму, когда массив разделяется на k подмассивов. Алгоритм использует двоичный поиск для сужения диапазона возможных сумм и проверяет допустимость каждого значения с помощью жадного метода, что приводит к эффективному решению задачи с временем работы **O(n log(sum))**.

1. **My code :** https://github.com/RomDhan19/polygon/tree/lab6/lab6-410

Таблица 1 - Результаты тестирования алгоритма

|  |  |  |
| --- | --- | --- |
| Имя | № Студента | IQ |
| Torchi Romdhan | 467746 |  |