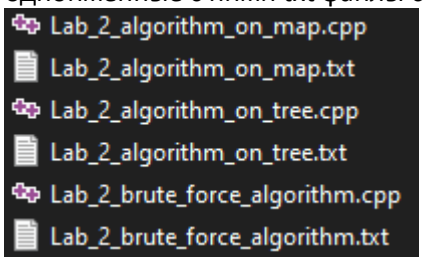
Отчет по лабораторной №2

Выполнил: Волков Владислав Сергеевич

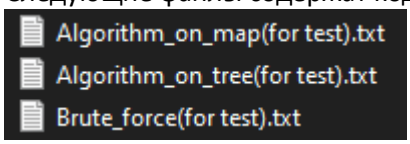
Группа: 21ПИ-3

В контесте использовались два логина:
vsvolkov@edu.hse.ru – с этого аккаунта отправлен только 3 алгоритм
Влад Волков – с этого аккаунта проверил все алгоритмы на корректность

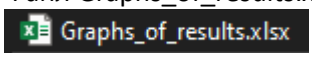
Для реализации алгоритмов использовалась: Visual Studio 2022 (C++)
Файлы формата .cpp содержат исключительно сами алгоритмы,
одноименные с ними txt файлы содержат тот же код в текстовом виде:



Следующие файлы содержат код, используемый для подсчета затрачиваемого времени на тестах:



Файл Graphs_of_results.xlsx содержит таблицу и графики полученных результатов:



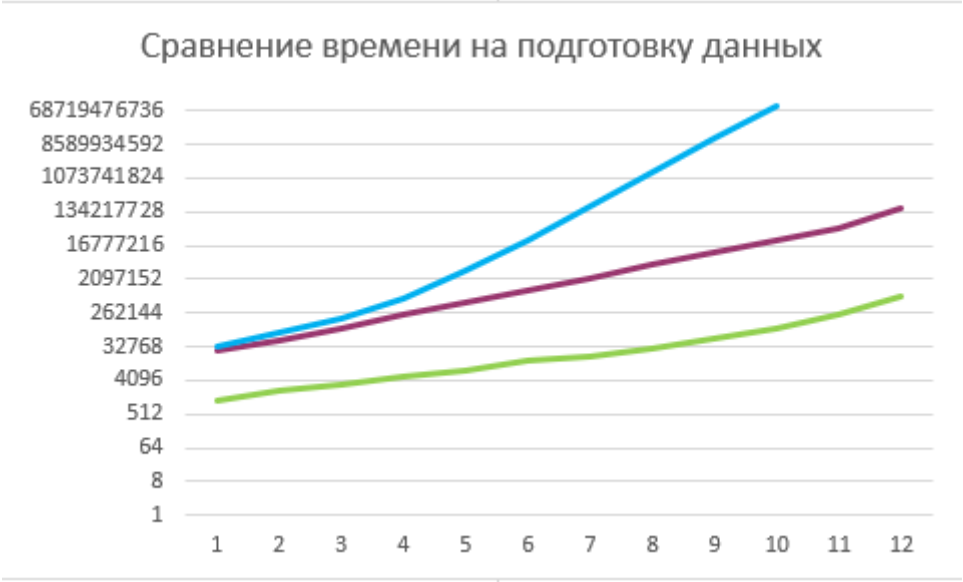
В качестве тестового набора использовал набор вложенных друг-в-друга с координатами {(10*i, 10*i), (10*(2N-i), 10*(2N-i))}.

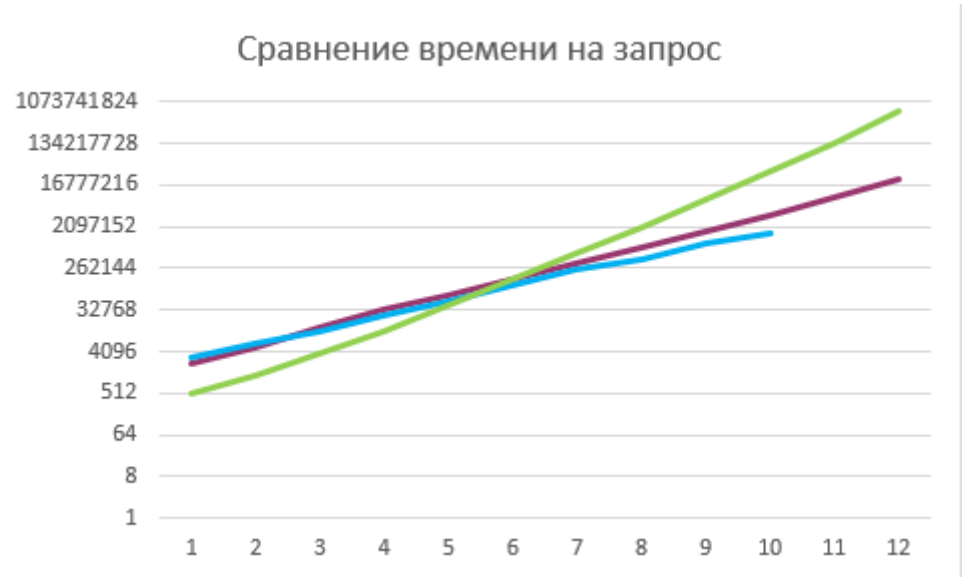
Для каждого набора делалось 4 * N запросов, точки, находящиеся на диагонали (0,0) – (20*N, 20*N) с шагом 5.

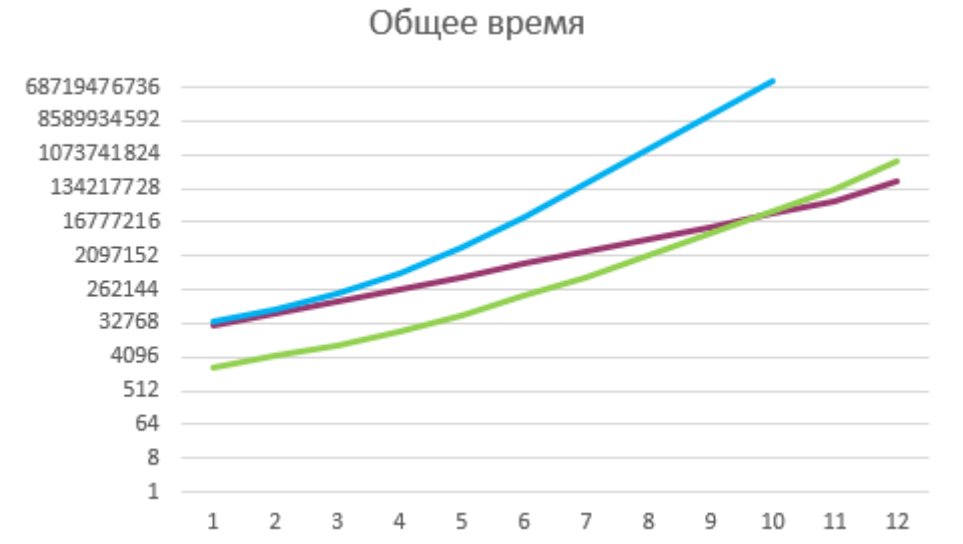
N: {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048}

Algorithm \ Rectangles(number)		2	4	8	16	32	64	128	256	512	1024	2048	4096
Brute force (aka полный перебор)	Среднее время подготовки(~) в наносекундах	1170	2150	3357	5122	7857	14428	18056	30606	57698	100406	254297	738788
	Среднее время запроса(~) в наносекундах	518	1262	3745	11701	39616	147460	544164	2117707	8432288	33180798	133643297	697801805
	Среднее время(общее)(~) в наносекундах	2172	4325	8806	20049	54143	175410	587733	2199353	8594941	33487631	134340390	700735134
Algorithm on map	Среднее время подготовки(~) в наносекундах	33164	74304	180723	660091	3625687	23629877	179377264	1,508E+09	1,2E+10	9,54E+10	Слишком долго идет подготовка, >10 минут	
	Среднее время запроса(~) в наносекундах	2936	5925	11549	24340	51900	109449	243103	401350	892350	1583900		
	Среднее время(общее)(~) в наносекундах	36661	81224	194090	687876	3684391	23752753	179647481	1,509E+09	1,2E+10	9,54E+10		
Algorithm on segment tree	Среднее время подготовки(~) в наносекундах	25144	50964	106948	229524	500845	1086105	2312670	5066716	10983334	23488799	51345545	173495271
	Среднее время запроса(~) в наносекундах	2150	5213	14045	32291	69775	157643	351664	771685	1708547	3907844	8758804	23158126
	Среднее время(общее)(~) в наносекундах	27815	57128	123046	265251	577453	1257357	2690952	5892295	12798214	27612626	60531665	197662129

По полученным данным получились следующие графики:

1. 

2. 

3. 

Выводы:

1. Подготовка данных

На основании полученных данных можно сделать вывод, что первому алгоритму практически не требуется подготовка, кроме как считывания координат прямоугольников, что происходит за константное время.

Второму алгоритму требуется существенно больше времени на подготовку, особенно при больших данных, так как мы строим огромную карту(в худшем случае N x N, если все координаты оказались уникальны), и из-за заполнения такой карты полным перебором координат(даже учитывая, что они сжатые) мы получаем “ужасный” O(N^3) в худшем случае.

Третий алгоритм же, в свою очередь работает существенно быстрее чем второй, но все же уступает первому, но это компенсируется при выполнении запросов, о чем в следующем пункте. Сложность такого построения будет O(N*logN), так как этот алгоритм реализуется через персистентное дерево отрезков, в реализации которого используется сканирующая прямая, которой мы проходим по уникальным x(или y, зависит от конкретной реализации), и добавляем в наше дерево отрезков модификаторы, процесс добавления занимает O(logN), так как это высота нашего дерева, а x(или y) у нас в худшем случае N, отсюда и сложность O(N*logN)

2. Запросы

В первом алгоритме все просто – идем по всем прямоугольникам и считаем попадает ли координата или нет, отсюда – O(N), сложность вроде бы линейная, но при большом количестве прямоугольников она начинает резко уступать двум другим алгоритмам.

Во втором – O(log N) за счет бинарного поиска, которым мы ищем позицию элемента в нашей карте, это уже гораздо быстрее, но этот алгоритм слишком сильно проигрывает в подготовке данных.

В третьем – также O(logN) за счет поиска позиции бинарным поиском, но здесь будет слегка больше времени, так как нам нужно еще пройтись по дереву, но это все еще logN, только с большей константой

3. Итог

Для простых задач, требующих частых добавлений данных, но при этом редких запросах, можно использовать полный перебор. Однако, задача в первую очередь состоит в том, чтобы быстрее выдавать ответ на запрос. Этот алгоритм достаточно плохой с точки зрения запросов, но вот он практически не требует подготовки данных, в чем его преимущество.

Второй алгоритм требует меньше времени на запрос, но тратит огромное количество времени на подготовку. Хорош тем, что ответ на запрос занимает в чистом виде O(log N).

Третий алгоритм самый “сбалансированный’ подготовка за N*logN и поиск за logN, хотя с некоторой константой, но как мы видим на графике, не сильно уступает алгоритму на карте.