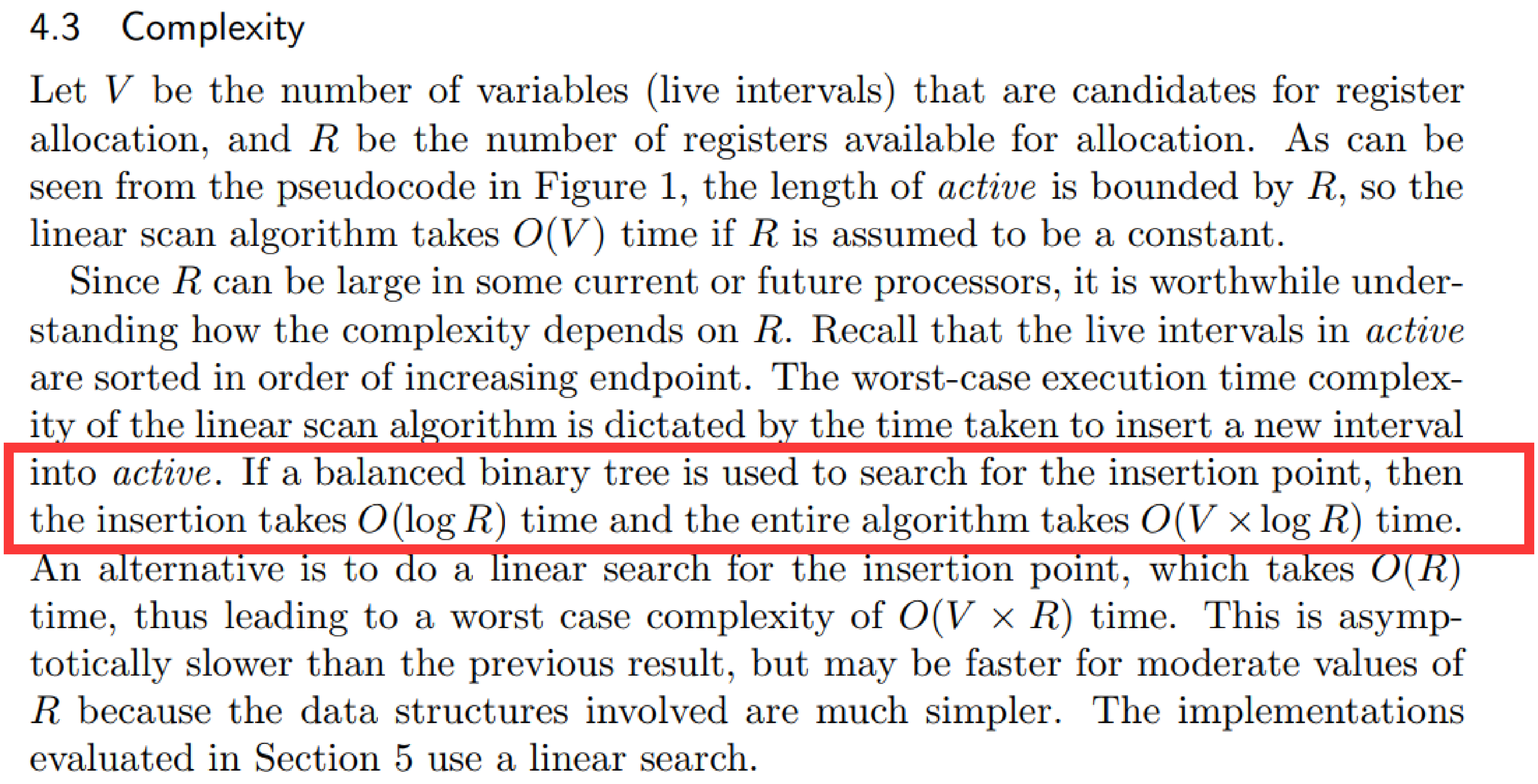
Introduction

我根据自己的理解用C++实现的一个线性分配算法
利用`std::multiset`作为平衡树，该算法的时间复杂度为 $\mathcal{O}(n\log R)$，n 为活跃区间数量，R 为寄存器的数量
该算法相比原始算法有如下改进：

1. 任意时刻寄存器空出来了都会尝试重新分配，算法尽可能让寄存器得到最充分的利用
2. 同一个变量可以输入多个活跃区间，以获得更精细的分配策略（原始算法仅仅是找一个最小的活跃区间覆盖所有活跃信息）
3. 使用STL的容器，将时间复杂度由 $\mathcal{O}(nR)$ 降低为 $\mathcal{O}(n\log R)$（尽管大多数情况下寄存器比较少，在实际中可能提升不大）



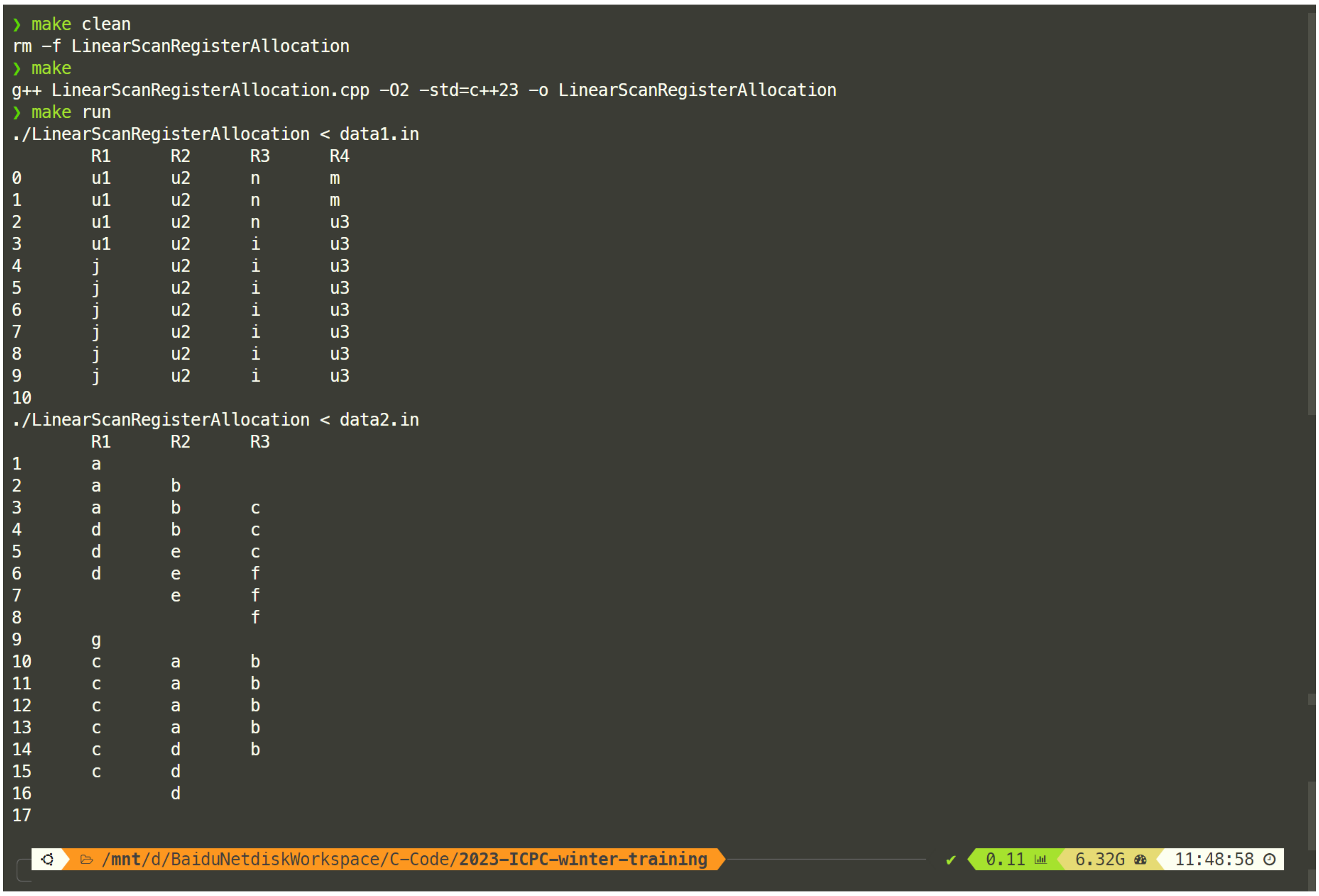
4.3 Complexity

Let V be the number of variables (live intervals) that are candidates for register allocation, and R be the number of registers available for allocation. As can be seen from the pseudocode in Figure 1, the length of *active* is bounded by R, so the linear scan algorithm takes $O(V)$ time if R is assumed to be a constant.

Since R can be large in some current or future processors, it is worthwhile understanding how the complexity depends on R. Recall that the live intervals in *active* are sorted in order of increasing endpoint. The worst-case execution time complexity of the linear scan algorithm is dictated by the time taken to insert a new interval into *active*. If a balanced binary tree is used to search for the insertion point, then the insertion takes $O(\log R)$ time and the entire algorithm takes $O(V \times \log R)$ time. An alternative is to do a linear search for the insertion point, which takes $O(R)$ time, thus leading to a worst case complexity of $O(V \times R)$ time. This is asymptotically slower than the previous result, but may be faster for moderate values of R because the data structures involved are much simpler. The implementations evaluated in Section 5 use a linear search.

Build && Run

```
make clean
make
make run
```



```
❯ make clean
rm -f LinearScanRegisterAllocation
❯ make
g++ LinearScanRegisterAllocation.cpp -O2 -std=c++23 -o LinearScanRegisterAllocation
❯ make run
./LinearScanRegisterAllocation < data1.in
        R1      R2      R3      R4
0       u1      u2      n       m
1       u1      u2      n       m
2       u1      u2      n       u3
3       u1      u2      i       u3
4       j       u2      i       u3
5       j       u2      i       u3
6       j       u2      i       u3
7       j       u2      i       u3
8       j       u2      i       u3
9       j       u2      i       u3
10
./LinearScanRegisterAllocation < data2.in
        R1      R2      R3
1       a
2       a       b
3       a       b       c
4       d       b       c
5       d       e       c
6       d       e       f
7               e       f
8                       f
9       g
10      c       a       b
11      c       a       b
12      c       a       b
13      c       a       b
14      c       d       b
15      c       d
16              d
17
  /mnt/d/BaiduNetdiskWorkspace/C-Code/2023-ICPC-winter-training          ✔  0.11   6.32G   11:48:58
```