project 2

● Graded

Group

Hrushik Chiluvuri
Jiyong Kwag
Nirmal Philipose Mathew
View or edit group

Total Points

14 / 18 pts

Autograder Score

14.0 / 18.0

Failed Tests

Test the band_join (0/2)

Test the band_join with SIMD (0/2)

Passed Tests

Test the low bin search and it should always pass (2/2)

Compilation with mavx2 (2/2)

Compilation with mavx512f (2/2)

Test the low_bin_nb_arithmetic (2/2)

Test the low_bin_nb_mask (2/2)

Test the low_bin_nb_4x (2/2)

Test the low_bin_nb_simd (2/2)

Question 2

Autograder's points is just for reference and it does not convert to the points on Carmen **0** / 0 pts

✓ **– 0 pts** Correct

Autograder Results

Autograder Output (hidden from students)

```
Test band join
inner[0] = 0
inner[1] = 2
inner[2] = 4
inner[3] = 6
inner[4] = 8
inner[5] = 10
inner[6] = 12
inner[7] = 14
inner[8] = 16
inner[9] = 18
inner[10] = 20
inner[11] = 22
inner[12] = 24
inner[13] = 26
inner[14] = 28
inner[15] = 30
inner[16] = 32
inner[17] = 34
inner[18] = 36
inner[19] = 38
inner[20] = 40
inner[21] = 42
inner[22] = 44
inner[23] = 46
inner[24] = 48
inner[25] = 50
inner[26] = 52
inner[27] = 54
inner[28] = 56
inner[29] = 58
inner[30] = 60
inner[31] = 62
inner[32] = 64
inner[33] = 66
inner[34] = 68
inner[35] = 70
inner[36] = 72
inner[37] = 74
inner[38] = 76
inner[39] = 78
inner[40] = 80
inner[41] = 82
inner[42] = 84
inner[43] = 86
inner[44] = 88
inner[45] = 90
inner[46] = 92
```

```
inner[47] = 94
inner[48] = 96
inner[49] = 98
inner[50] = 100
inner[51] = 102
inner[52] = 104
inner[53] = 106
inner[54] = 108
inner[55] = 110
inner[56] = 112
inner[57] = 114
inner[58] = 116
inner[59] = 118
inner[60] = 120
inner[61] = 122
inner[62] = 124
inner[63] = 126
inner[64] = 128
inner[65] = 130
inner[66] = 132
inner[67] = 134
inner[68] = 136
inner[69] = 138
inner[70] = 140
inner[71] = 142
inner[72] = 144
inner[73] = 146
inner[74] = 148
inner[75] = 150
inner[76] = 152
inner[77] = 154
inner[78] = 156
inner[79] = 158
inner[80] = 160
inner[81] = 162
inner[82] = 164
inner[83] = 166
inner[84] = 168
inner[85] = 170
inner[86] = 172
inner[87] = 174
inner[88] = 176
inner[89] = 178
inner[90] = 180
inner[91] = 182
inner[92] = 184
inner[93] = 186
inner[94] = 188
inner[95] = 190
```

```
inner[96] = 192
inner[97] = 194
inner[98] = 196
inner[99] = 198
outer[0] = 80
outer[1] = 78
outer[2] = 76
outer[3] = 74
outer[4] = 72
outer[5] = 70
outer[6] = 68
outer[7] = 66
outer[8] = 64
outer[9] = 62
outer[10] = 60
outer[11] = 58
outer[12] = 56
outer[13] = 54
outer[14] = 52
outer[15] = 50
outer[16] = 48
outer[17] = 46
outer[18] = 44
outer[19] = 42
outer[20] = 40
outer[21] = 38
outer[22] = 36
outer[23] = 34
outer[24] = 32
outer[25] = 30
outer[26] = 28
outer[27] = 26
outer[28] = 24
outer[29] = 22
outer[30] = 20
outer[31] = 18
outer[32] = 16
outer[33] = 14
outer[34] = 12
outer[35] = 10
outer[36] = 8
outer[37] = 6
outer[38] = 4

band join with band 4 return 156 tuples
The return number 156 is incorrect
( 0, 38)
( 1, 37)
( 1, 38)
( 2, 36)
( 2, 37)
( 2, 38)
( 3, 35)
( 3, 36)
( 3, 37)
( 3, 38)
( 4, 34)
( 4, 35)
( 4, 36)
( 4, 37)
( 5, 33)
( 5, 34)
( 5, 35)
( 5, 36)
( 6, 32)
( 6, 33)
( 6, 34)
( 6, 35)
( 7, 31)
( 7, 32)
( 7, 33)
( 7, 34)
( 8, 30)
( 8, 31)
( 8, 32)
( 8, 33)
( 9, 29)
( 9, 30)
( 9, 31)
( 9, 32)
( 10, 28)
( 10, 29)
( 10, 30)
( 10, 31)
( 11, 27)
( 11, 28)
( 11, 29)
( 11, 30)
( 12, 26)
( 12, 27)
( 12, 28)
( 12, 29)
( 13, 25)
( 13, 26)
( 13, 27)
( 13, 28)
( 14, 24)
( 14, 25)
( 14, 26)
( 14, 27)
( 15, 23)
( 15, 24)
( 15, 25)
( 15, 26)
( 16, 22)
( 16, 23)
( 16, 24)
( 16, 25)
( 17, 21)
( 17, 22)
( 17, 23)
( 17, 24)
( 18, 20)
( 18, 21)
( 18, 22)
( 18, 23)
( 19, 19)
( 19, 20)
( 19, 21)
( 19, 22)
( 20, 18)
( 20, 19)
( 20, 20)
( 20, 21)
( 21, 17)
( 21, 18)
( 21, 19)
( 21, 20)
( 22, 16)
( 22, 17)
( 22, 18)
( 22, 19)
( 23, 15)
( 23, 16)
( 23, 17)
( 23, 18)
( 24, 14)
( 24, 15)
( 24, 16)
( 24, 17)
( 25, 13)
( 25, 14)
( 25, 15)
( 25, 16)
( 26, 12)
( 26, 13)
( 26, 14)
( 26, 15)
( 27, 11)
( 27, 12)
( 27, 13)
( 27, 14)
( 28, 10)
( 28, 11)
( 28, 12)
( 28, 13)
( 29, 9)
( 29, 10)
( 29, 11)
( 29, 12)
( 30, 8)
( 30, 9)
( 30, 10)
( 30, 11)
( 31, 7)
( 31, 8)
( 31, 9)
( 31, 10)
( 32, 6)
( 32, 7)
( 32, 8)
( 32, 9)
( 33, 5)
( 33, 6)
( 33, 7)
( 33, 8)
( 34, 4)
( 34, 5)
( 34, 6)
( 34, 7)
( 35, 3)
( 35, 4)
( 35, 5)
( 35, 6)
( 36, 2)
( 36, 3)
( 36, 4)
( 36, 5)
( 37, 1)
( 37, 2)
( 37, 3)
( 37, 4)
( 38, 0)
( 38, 1)
( 38, 2)
( 38, 3)
( 39, 0)
( 39, 1)
( 39, 2)
( 40, 0)
( 40, 1)
( 41, 0)
Some error in the band join, correct result should be
(0, 38)
(1, 37)
(1, 38)
(2, 36)
(2, 37)
(2, 38)
(3, 35)
(3, 36)
(3, 37)
(3, 38)
(4, 34)
(4, 35)
(4, 36)
(4, 37)
(4, 38)
(5, 33)
(5, 34)
(5, 35)
(5, 36)
(5, 37)
(6, 32)
(6, 33)
(6, 34)
(6, 35)
(6, 36)
(7, 31)
(7, 32)
(7, 33)
(7, 34)
(7, 35)
(8, 30)
(8, 31)
(8, 32)
(8, 33)
(8, 34)
(9, 29)
(9, 30)
(9, 31)
(9, 32)
(9, 33)
(10, 28)
(10, 29)
```

```
(10, 30)
(10, 31)
(10, 32)
(11, 27)
(11, 28)
(11, 29)
(11, 30)
(11, 31)
(12, 26)
(12, 27)
(12, 28)
(12, 29)
(12, 30)
(13, 25)
(13, 26)
(13, 27)
(13, 28)
(13, 29)
(14, 24)
(14, 25)
(14, 26)
(14, 27)
(14, 28)
(15, 23)
(15, 24)
(15, 25)
(15, 26)
(15, 27)
(16, 22)
(16, 23)
(16, 24)
(16, 25)
(16, 26)
(17, 21)
(17, 22)
(17, 23)
(17, 24)
(17, 25)
(18, 20)
(18, 21)
(18, 22)
(18, 23)
(18, 24)
(19, 19)
(19, 20)
(19, 21)
(19, 22)
(19, 23)
(20, 18)
```

```
(20, 19)
(20, 20)
(20, 21)
(20, 22)
(21, 17)
(21, 18)
(21, 19)
(21, 20)
(21, 21)
(22, 16)
(22, 17)
(22, 18)
(22, 19)
(22, 20)
(23, 15)
(23, 16)
(23, 17)
(23, 18)
(23, 19)
(24, 14)
(24, 15)
(24, 16)
(24, 17)
(24, 18)
(25, 13)
(25, 14)
(25, 15)
(25, 16)
(25, 17)
(26, 12)
(26, 13)
(26, 14)
(26, 15)
(26, 16)
(27, 11)
(27, 12)
(27, 13)
(27, 14)
(27, 15)
(28, 10)
(28, 11)
(28, 12)
(28, 13)
(28, 14)
(29, 9)
(29, 10)
(29, 11)
(29, 12)
(29, 13)
```

```
(30, 8)
(30, 9)
(30, 10)
(30, 11)
(30, 12)
(31, 7)
(31, 8)
(31, 9)
(31, 10)
(31, 11)
(32, 6)
(32, 7)
(32, 8)
(32, 9)
(32, 10)
(33, 5)
(33, 6)
(33, 7)
(33, 8)
(33, 9)
(34, 4)
(34, 5)
(34, 6)
(34, 7)
(34, 8)
(35, 3)
(35, 4)
(35, 5)
(35, 6)
(35, 7)
(36, 2)
(36, 3)
(36, 4)
(36, 5)
(36, 6)
(37, 1)
(37, 2)
(37, 3)
(37, 4)
(37, 5)
(38, 0)
(38, 1)
(38, 2)
(38, 3)
(38, 4)
(39, 0)
(39, 1)
(39, 2)
(39, 3)
```

```
(40, 0)
(40, 1)
(40, 2)
(41, 0)
(41, 1)
(42, 0)
Some error happens, will try with a reverse order
inner[0] = 198
inner[1] = 196
inner[2] = 194
inner[3] = 192
inner[4] = 190
inner[5] = 188
inner[6] = 186
inner[7] = 184
inner[8] = 182
inner[9] = 180
inner[10] = 178
inner[11] = 176
inner[12] = 174
inner[13] = 172
inner[14] = 170
inner[15] = 168
inner[16] = 166
inner[17] = 164
inner[18] = 162
inner[19] = 160
inner[20] = 158
inner[21] = 156
inner[22] = 154
inner[23] = 152
inner[24] = 150
inner[25] = 148
inner[26] = 146
inner[27] = 144
inner[28] = 142
inner[29] = 140
inner[30] = 138
inner[31] = 136
inner[32] = 134
inner[33] = 132
inner[34] = 130
inner[35] = 128
inner[36] = 126
inner[37] = 124
inner[38] = 122
inner[39] = 120
inner[40] = 118
inner[41] = 116
```

```
inner[42] = 114
inner[43] = 112
inner[44] = 110
inner[45] = 108
inner[46] = 106
inner[47] = 104
inner[48] = 102
inner[49] = 100
inner[50] = 98
inner[51] = 96
inner[52] = 94
inner[53] = 92
inner[54] = 90
inner[55] = 88
inner[56] = 86
inner[57] = 84
inner[58] = 82
inner[59] = 80
inner[60] = 78
inner[61] = 76
inner[62] = 74
inner[63] = 72
inner[64] = 70
inner[65] = 68
inner[66] = 66
inner[67] = 64
inner[68] = 62
inner[69] = 60
inner[70] = 58
inner[71] = 56
inner[72] = 54
inner[73] = 52
inner[74] = 50
inner[75] = 48
inner[76] = 46
inner[77] = 44
inner[78] = 42
inner[79] = 40
inner[80] = 38
inner[81] = 36
inner[82] = 34
inner[83] = 32
inner[84] = 30
inner[85] = 28
inner[86] = 26
inner[87] = 24
inner[88] = 22
inner[89] = 20
inner[90] = 18
```

```
inner[91] = 16
inner[92] = 14
inner[93] = 12
inner[94] = 10
inner[95] = 8
inner[96] = 6
inner[97] = 4
inner[98] = 2
inner[99] = 0
outer[0] = 4
outer[1] = 6
outer[2] = 8
outer[3] = 10
outer[4] = 12
outer[5] = 14
outer[6] = 16
outer[7] = 18
outer[8] = 20
outer[9] = 22
outer[10] = 24
outer[11] = 26
outer[12] = 28
outer[13] = 30
outer[14] = 32
outer[15] = 34
outer[16] = 36
outer[17] = 38
outer[18] = 40
outer[19] = 42
outer[20] = 44
outer[21] = 46
outer[22] = 48
outer[23] = 50
outer[24] = 52
outer[25] = 54
outer[26] = 56
outer[27] = 58
outer[28] = 60
outer[29] = 62
outer[30] = 64
outer[31] = 66
outer[32] = 68
outer[33] = 70
outer[34] = 72
outer[35] = 74
outer[36] = 76
outer[37] = 78
outer[38] = 80
```

```
band join with band 4 return 0 tuples
The return number 0 is incorrect
Some error in the band join, correct result should be
(57, 38)
(58, 37)
(58, 38)
(59, 36)
(59, 37)
(59, 38)
(60, 35)
(60, 36)
(60, 37)
(60, 38)
(61, 34)
(61, 35)
(61, 36)
(61, 37)
(61, 38)
(62, 33)
(62, 34)
(62, 35)
(62, 36)
(62, 37)
(63, 32)
(63, 33)
(63, 34)
(63, 35)
(63, 36)
(64, 31)
(64, 32)
(64, 33)
(64, 34)
(64, 35)
(65, 30)
(65, 31)
(65, 32)
(65, 33)
(65, 34)
(66, 29)
(66, 30)
(66, 31)
(66, 32)
(66, 33)
(67, 28)
(67, 29)
(67, 30)
(67, 31)
(67, 32)
(68, 27)
```

```
(68, 28)
(68, 29)
(68, 30)
(68, 31)
(69, 26)
(69, 27)
(69, 28)
(69, 29)
(69, 30)
(70, 25)
(70, 26)
(70, 27)
(70, 28)
(70, 29)
(71, 24)
(71, 25)
(71, 26)
(71, 27)
(71, 28)
(72, 23)
(72, 24)
(72, 25)
(72, 26)
(72, 27)
(73, 22)
(73, 23)
(73, 24)
(73, 25)
(73, 26)
(74, 21)
(74, 22)
(74, 23)
(74, 24)
(74, 25)
(75, 20)
(75, 21)
(75, 22)
(75, 23)
(75, 24)
(76, 19)
(76, 20)
(76, 21)
(76, 22)
(76, 23)
(77, 18)
(77, 19)
(77, 20)
(77, 21)
(77, 22)
```

```
(78, 17)
(78, 18)
(78, 19)
(78, 20)
(78, 21)
(79, 16)
(79, 17)
(79, 18)
(79, 19)
(79, 20)
(80, 15)
(80, 16)
(80, 17)
(80, 18)
(80, 19)
(81, 14)
(81, 15)
(81, 16)
(81, 17)
(81, 18)
(82, 13)
(82, 14)
(82, 15)
(82, 16)
(82, 17)
(83, 12)
(83, 13)
(83, 14)
(83, 15)
(83, 16)
(84, 11)
(84, 12)
(84, 13)
(84, 14)
(84, 15)
(85, 10)
(85, 11)
(85, 12)
(85, 13)
(85, 14)
(86, 9)
(86, 10)
(86, 11)
(86, 12)
(86, 13)
(87, 8)
(87, 9)
(87, 10)
(87, 11)
```

```
(87, 12)
(88, 7)
(88, 8)
(88, 9)
(88, 10)
(88, 11)
(89, 6)
(89, 7)
(89, 8)
(89, 9)
(89, 10)
(90, 5)
(90, 6)
(90, 7)
(90, 8)
(90, 9)
(91, 4)
(91, 5)
(91, 6)
(91, 7)
(91, 8)
(92, 3)
(92, 4)
(92, 5)
(92, 6)
(92, 7)
(93, 2)
(93, 3)
(93, 4)
(93, 5)
(93, 6)
(94, 1)
(94, 2)
(94, 3)
(94, 4)
(94, 5)
(95, 0)
(95, 1)
(95, 2)
(95, 3)
(95, 4)
(96, 0)
(96, 1)
(96, 2)
(96, 3)
(97, 0)
(97, 1)
(97, 2)
(98, 0)
```

```
(98, 1)
(99, 0)
Test band join with SIMD
inner[0] = 0
inner[1] = 2
inner[2] = 4
inner[3] = 6
inner[4] = 8
inner[5] = 10
inner[6] = 12
inner[7] = 14
inner[8] = 16
inner[9] = 18
inner[10] = 20
inner[11] = 22
inner[12] = 24
inner[13] = 26
inner[14] = 28
inner[15] = 30
inner[16] = 32
inner[17] = 34
inner[18] = 36
inner[19] = 38
inner[20] = 40
inner[21] = 42
inner[22] = 44
inner[23] = 46
inner[24] = 48
inner[25] = 50
inner[26] = 52
inner[27] = 54
inner[28] = 56
inner[29] = 58
inner[30] = 60
inner[31] = 62
inner[32] = 64
inner[33] = 66
inner[34] = 68
inner[35] = 70
inner[36] = 72
inner[37] = 74
inner[38] = 76
inner[39] = 78
inner[40] = 80
inner[41] = 82
inner[42] = 84
inner[43] = 86
inner[44] = 88
inner[45] = 90
```

```
inner[46] = 92
inner[47] = 94
inner[48] = 96
inner[49] = 98
inner[50] = 100
inner[51] = 102
inner[52] = 104
inner[53] = 106
inner[54] = 108
inner[55] = 110
inner[56] = 112
inner[57] = 114
inner[58] = 116
inner[59] = 118
inner[60] = 120
inner[61] = 122
inner[62] = 124
inner[63] = 126
inner[64] = 128
inner[65] = 130
inner[66] = 132
inner[67] = 134
inner[68] = 136
inner[69] = 138
inner[70] = 140
inner[71] = 142
inner[72] = 144
inner[73] = 146
inner[74] = 148
inner[75] = 150
inner[76] = 152
inner[77] = 154
inner[78] = 156
inner[79] = 158
inner[80] = 160
inner[81] = 162
inner[82] = 164
inner[83] = 166
inner[84] = 168
inner[85] = 170
inner[86] = 172
inner[87] = 174
inner[88] = 176
inner[89] = 178
inner[90] = 180
inner[91] = 182
inner[92] = 184
inner[93] = 186
inner[94] = 188
```

```
inner[95] = 190
inner[96] = 192
inner[97] = 194
inner[98] = 196
inner[99] = 198
outer[0] = 80
outer[1] = 78
outer[2] = 76
outer[3] = 74
outer[4] = 72
outer[5] = 70
outer[6] = 68
outer[7] = 66
outer[8] = 64
outer[9] = 62
outer[10] = 60
outer[11] = 58
outer[12] = 56
outer[13] = 54
outer[14] = 52
outer[15] = 50
outer[16] = 48
outer[17] = 46
outer[18] = 44
outer[19] = 42
outer[20] = 40
outer[21] = 38
outer[22] = 36
outer[23] = 34
outer[24] = 32
outer[25] = 30
outer[26] = 28
outer[27] = 26
outer[28] = 24
outer[29] = 22
outer[30] = 20
outer[31] = 18
outer[32] = 16
outer[33] = 14
outer[34] = 12
outer[35] = 10
outer[36] = 8
outer[37] = 6
outer[38] = 4
band join with band 4 return 156 tuples
The return number 156 is incorrect
( 0, 38)
( 1, 37)
( 1, 38)
( 2, 36)
( 2, 37)
( 2, 38)
( 3, 35)
( 3, 36)
( 3, 37)
( 3, 38)
( 4, 34)
( 4, 35)
( 4, 36)
( 4, 37)
( 5, 33)
( 5, 34)
( 5, 35)
( 5, 36)
( 6, 32)
( 6, 33)
( 6, 34)
( 6, 35)
( 7, 31)
( 7, 32)
( 7, 33)
( 7, 34)
( 8, 30)
( 8, 31)
( 8, 32)
( 8, 33)
( 9, 29)
( 9, 30)
( 9, 31)
( 9, 32)
( 10, 28)
( 10, 29)
( 10, 30)
( 10, 31)
( 11, 27)
( 11, 28)
( 11, 29)
( 11, 30)
( 12, 26)
( 12, 27)
( 12, 28)
( 12, 29)
( 13, 25)
( 13, 26)
( 13, 27)
( 13, 28)
( 14, 24)
( 14, 25)
( 14, 26)
( 14, 27)
( 15, 23)
( 15, 24)
( 15, 25)
( 15, 26)
( 16, 22)
( 16, 23)
( 16, 24)
( 16, 25)
( 17, 21)
( 17, 22)
( 17, 23)
( 17, 24)
( 18, 20)
( 18, 21)
( 18, 22)
( 18, 23)
( 19, 19)
( 19, 20)
( 19, 21)
( 19, 22)
( 20, 18)
( 20, 19)
( 20, 20)
( 20, 21)
( 21, 17)
( 21, 18)
( 21, 19)
( 21, 20)
( 22, 16)
( 22, 17)
( 22, 18)
( 22, 19)
( 23, 15)
( 23, 16)
( 23, 17)
( 23, 18)
( 24, 14)
( 24, 15)
( 24, 16)
( 24, 17)
( 25, 13)
( 25, 14)
( 25, 15)
( 25, 16)
( 26, 12)
( 26, 13)
( 26, 14)
( 26, 15)
( 27, 11)
( 27, 12)
( 27, 13)
( 27, 14)
( 28, 10)
( 28, 11)
( 28, 12)
( 28, 13)
( 29, 9)
( 29, 10)
( 29, 11)
( 29, 12)
( 30, 8)
( 30, 9)
( 30, 10)
( 30, 11)
( 31, 7)
( 31, 8)
( 31, 9)
( 31, 10)
( 32, 6)
( 32, 7)
( 32, 8)
( 32, 9)
( 33, 5)
( 33, 6)
( 33, 7)
( 33, 8)
( 34, 4)
( 34, 5)
( 34, 6)
( 34, 7)
( 35, 3)
( 35, 4)
( 35, 5)
( 35, 6)
( 36, 2)
( 36, 3)
( 36, 4)
( 36, 5)
( 37, 1)
( 37, 2)
( 37, 3)
( 37, 4)
( 38, 0)
( 38, 1)
( 38, 2)
( 38, 3)
( 39, 0)
( 39, 1)
( 39, 2)
( 40, 0)
( 40, 1)
( 41, 0)
Some error in the band join, correct result should be
(0, 38)
(1, 37)
(1, 38)
(2, 36)
(2, 37)
(2, 38)
(3, 35)
(3, 36)
(3, 37)
(3, 38)
(4, 34)
(4, 35)
(4, 36)
(4, 37)
(4, 38)
(5, 33)
(5, 34)
(5, 35)
(5, 36)
(5, 37)
(6, 32)
(6, 33)
(6, 34)
(6, 35)
(6, 36)
(7, 31)
(7, 32)
(7, 33)
(7, 34)
(7, 35)
(8, 30)
(8, 31)
(8, 32)
(8, 33)
(8, 34)
(9, 29)
(9, 30)
(9, 31)
(9, 32)
(9, 33)
(10, 28)
(10, 29)
```

```
(10, 30)
(10, 31)
(10, 32)
(11, 27)
(11, 28)
(11, 29)
(11, 30)
(11, 31)
(12, 26)
(12, 27)
(12, 28)
(12, 29)
(12, 30)
(13, 25)
(13, 26)
(13, 27)
(13, 28)
(13, 29)
(14, 24)
(14, 25)
(14, 26)
(14, 27)
(14, 28)
(15, 23)
(15, 24)
(15, 25)
(15, 26)
(15, 27)
(16, 22)
(16, 23)
(16, 24)
(16, 25)
(16, 26)
(17, 21)
(17, 22)
(17, 23)
(17, 24)
(17, 25)
(18, 20)
(18, 21)
(18, 22)
(18, 23)
(18, 24)
(19, 19)
(19, 20)
(19, 21)
(19, 22)
(19, 23)
(20, 18)
```

```
(20, 19)
(20, 20)
(20, 21)
(20, 22)
(21, 17)
(21, 18)
(21, 19)
(21, 20)
(21, 21)
(22, 16)
(22, 17)
(22, 18)
(22, 19)
(22, 20)
(23, 15)
(23, 16)
(23, 17)
(23, 18)
(23, 19)
(24, 14)
(24, 15)
(24, 16)
(24, 17)
(24, 18)
(25, 13)
(25, 14)
(25, 15)
(25, 16)
(25, 17)
(26, 12)
(26, 13)
(26, 14)
(26, 15)
(26, 16)
(27, 11)
(27, 12)
(27, 13)
(27, 14)
(27, 15)
(28, 10)
(28, 11)
(28, 12)
(28, 13)
(28, 14)
(29, 9)
(29, 10)
(29, 11)
(29, 12)
(29, 13)
```

```
(30, 8)
(30, 9)
(30, 10)
(30, 11)
(30, 12)
(31, 7)
(31, 8)
(31, 9)
(31, 10)
(31, 11)
(32, 6)
(32, 7)
(32, 8)
(32, 9)
(32, 10)
(33, 5)
(33, 6)
(33, 7)
(33, 8)
(33, 9)
(34, 4)
(34, 5)
(34, 6)
(34, 7)
(34, 8)
(35, 3)
(35, 4)
(35, 5)
(35, 6)
(35, 7)
(36, 2)
(36, 3)
(36, 4)
(36, 5)
(36, 6)
(37, 1)
(37, 2)
(37, 3)
(37, 4)
(37, 5)
(38, 0)
(38, 1)
(38, 2)
(38, 3)
(38, 4)
(39, 0)
(39, 1)
(39, 2)
(39, 3)
```

```
(40, 0)
(40, 1)
(40, 2)
(41, 0)
(41, 1)
(42, 0)
Some error happens, will try with a reverse order
inner[0] = 198
inner[1] = 196
inner[2] = 194
inner[3] = 192
inner[4] = 190
inner[5] = 188
inner[6] = 186
inner[7] = 184
inner[8] = 182
inner[9] = 180
inner[10] = 178
inner[11] = 176
inner[12] = 174
inner[13] = 172
inner[14] = 170
inner[15] = 168
inner[16] = 166
inner[17] = 164
inner[18] = 162
inner[19] = 160
inner[20] = 158
inner[21] = 156
inner[22] = 154
inner[23] = 152
inner[24] = 150
inner[25] = 148
inner[26] = 146
inner[27] = 144
inner[28] = 142
inner[29] = 140
inner[30] = 138
inner[31] = 136
inner[32] = 134
inner[33] = 132
inner[34] = 130
inner[35] = 128
inner[36] = 126
inner[37] = 124
inner[38] = 122
inner[39] = 120
inner[40] = 118
inner[41] = 116
```

```
inner[42] = 114
inner[43] = 112
inner[44] = 110
inner[45] = 108
inner[46] = 106
inner[47] = 104
inner[48] = 102
inner[49] = 100
inner[50] = 98
inner[51] = 96
inner[52] = 94
inner[53] = 92
inner[54] = 90
inner[55] = 88
inner[56] = 86
inner[57] = 84
inner[58] = 82
inner[59] = 80
inner[60] = 78
inner[61] = 76
inner[62] = 74
inner[63] = 72
inner[64] = 70
inner[65] = 68
inner[66] = 66
inner[67] = 64
inner[68] = 62
inner[69] = 60
inner[70] = 58
inner[71] = 56
inner[72] = 54
inner[73] = 52
inner[74] = 50
inner[75] = 48
inner[76] = 46
inner[77] = 44
inner[78] = 42
inner[79] = 40
inner[80] = 38
inner[81] = 36
inner[82] = 34
inner[83] = 32
inner[84] = 30
inner[85] = 28
inner[86] = 26
inner[87] = 24
inner[88] = 22
inner[89] = 20
inner[90] = 18
```

```
inner[91] = 16
inner[92] = 14
inner[93] = 12
inner[94] = 10
inner[95] = 8
inner[96] = 6
inner[97] = 4
inner[98] = 2
inner[99] = 0
outer[0] = 4
outer[1] = 6
outer[2] = 8
outer[3] = 10
outer[4] = 12
outer[5] = 14
outer[6] = 16
outer[7] = 18
outer[8] = 20
outer[9] = 22
outer[10] = 24
outer[11] = 26
outer[12] = 28
outer[13] = 30
outer[14] = 32
outer[15] = 34
outer[16] = 36
outer[17] = 38
outer[18] = 40
outer[19] = 42
outer[20] = 44
outer[21] = 46
outer[22] = 48
outer[23] = 50
outer[24] = 52
outer[25] = 54
outer[26] = 56
outer[27] = 58
outer[28] = 60
outer[29] = 62
outer[30] = 64
outer[31] = 66
outer[32] = 68
outer[33] = 70
outer[34] = 72
outer[35] = 74
outer[36] = 76
outer[37] = 78
outer[38] = 80
band join with band 4 return 0 tuples
```

```
The return number 0 is incorrect
Some error in the band join, correct result should be
(57, 38)
(58, 37)
(58, 38)
(59, 36)
(59, 37)
(59, 38)
(60, 35)
(60, 36)
(60, 37)
(60, 38)
(61, 34)
(61, 35)
(61, 36)
(61, 37)
(61, 38)
(62, 33)
(62, 34)
(62, 35)
(62, 36)
(62, 37)
(63, 32)
(63, 33)
(63, 34)
(63, 35)
(63, 36)
(64, 31)
(64, 32)
(64, 33)
(64, 34)
(64, 35)
(65, 30)
(65, 31)
(65, 32)
(65, 33)
(65, 34)
(66, 29)
(66, 30)
(66, 31)
(66, 32)
(66, 33)
(67, 28)
(67, 29)
(67, 30)
(67, 31)
(67, 32)
(68, 27)
(68, 28)
```

```
(68, 29)
(68, 30)
(68, 31)
(69, 26)
(69, 27)
(69, 28)
(69, 29)
(69, 30)
(70, 25)
(70, 26)
(70, 27)
(70, 28)
(70, 29)
(71, 24)
(71, 25)
(71, 26)
(71, 27)
(71, 28)
(72, 23)
(72, 24)
(72, 25)
(72, 26)
(72, 27)
(73, 22)
(73, 23)
(73, 24)
(73, 25)
(73, 26)
(74, 21)
(74, 22)
(74, 23)
(74, 24)
(74, 25)
(75, 20)
(75, 21)
(75, 22)
(75, 23)
(75, 24)
(76, 19)
(76, 20)
(76, 21)
(76, 22)
(76, 23)
(77, 18)
(77, 19)
(77, 20)
(77, 21)
(77, 22)
(78, 17)
```

```
(78, 18)
(78, 19)
(78, 20)
(78, 21)
(79, 16)
(79, 17)
(79, 18)
(79, 19)
(79, 20)
(80, 15)
(80, 16)
(80, 17)
(80, 18)
(80, 19)
(81, 14)
(81, 15)
(81, 16)
(81, 17)
(81, 18)
(82, 13)
(82, 14)
(82, 15)
(82, 16)
(82, 17)
(83, 12)
(83, 13)
(83, 14)
(83, 15)
(83, 16)
(84, 11)
(84, 12)
(84, 13)
(84, 14)
(84, 15)
(85, 10)
(85, 11)
(85, 12)
(85, 13)
(85, 14)
(86, 9)
(86, 10)
(86, 11)
(86, 12)
(86, 13)
(87, 8)
(87, 9)
(87, 10)
(87, 11)
(87, 12)
```

```
(88, 7)
(88, 8)
(88, 9)
(88, 10)
(88, 11)
(89, 6)
(89, 7)
(89, 8)
(89, 9)
(89, 10)
(90, 5)
(90, 6)
(90, 7)
(90, 8)
(90, 9)
(91, 4)
(91, 5)
(91, 6)
(91, 7)
(91, 8)
(92, 3)
(92, 4)
(92, 5)
(92, 6)
(92, 7)
(93, 2)
(93, 3)
(93, 4)
(93, 5)
(93, 6)
(94, 1)
(94, 2)
(94, 3)
(94, 4)
(94, 5)
(95, 0)
(95, 1)
(95, 2)
(95, 3)
(95, 4)
(96, 0)
(96, 1)
(96, 2)
(96, 3)
(97, 0)
(97, 1)
(97, 2)
(98, 0)
(98, 1)
```

```
(99, 0)
Test low_bin_search
data[0] = 0
data[1] = 2
data[2] = 4
data[3] = 6
data[4] = 8
data[5] = 10
data[6] = 12
data[7] = 14
data[8] = 16
data[9] = 18
data[10] = 20
data[11] = 22
data[12] = 24
data[13] = 26
data[14] = 28
data[15] = 30
data[16] = 32
data[17] = 34
data[18] = 36
data[19] = 38
data[20] = 40
data[21] = 42
data[22] = 44
data[23] = 46
data[24] = 48
data[25] = 50
data[26] = 52
data[27] = 54
data[28] = 56
data[29] = 58
data[30] = 60
data[31] = 62
data[32] = 64
data[33] = 66
data[34] = 68
data[35] = 70
data[36] = 72
data[37] = 74
data[38] = 76
data[39] = 78
data[40] = 80
data[41] = 82
data[42] = 84
data[43] = 86
data[44] = 88
data[45] = 90
data[46] = 92
```

```
data[47] = 94
data[48] = 96
data[49] = 98
data[50] = 100
data[51] = 102
data[52] = 104
data[53] = 106
data[54] = 108
data[55] = 110
data[56] = 112
data[57] = 114
data[58] = 116
data[59] = 118
data[60] = 120
data[61] = 122
data[62] = 124
data[63] = 126
data[64] = 128
data[65] = 130
data[66] = 132
data[67] = 134
data[68] = 136
data[69] = 138
data[70] = 140
data[71] = 142
data[72] = 144
data[73] = 146
data[74] = 148
data[75] = 150
data[76] = 152
data[77] = 154
data[78] = 156
data[79] = 158
data[80] = 160
data[81] = 162
data[82] = 164
data[83] = 166
data[84] = 168
data[85] = 170
data[86] = 172
data[87] = 174
data[88] = 176
data[89] = 178
data[90] = 180
data[91] = 182
data[92] = 184
data[93] = 186
data[94] = 188
data[95] = 190
```

```
data[96] = 192
data[97] = 194
data[98] = 196
data[99] = 198
Low_bin_search
search 80  index 40
Low_bin_search
search 78  index 39
Low_bin_search
search 76  index 38
Low_bin_search
search 74  index 37
Low_bin_search
search 72  index 36
Low_bin_search
search 70  index 35
Low_bin_search
search 68  index 34
Low_bin_search
search 66  index 33
Low_bin_search
search 64  index 32
Low_bin_search
search 62  index 31
Low_bin_search
search 60  index 30
Low_bin_search
search 58  index 29
Low_bin_search
search 56  index 28
Low_bin_search
search 54  index 27
Low_bin_search
search 52  index 26
Low_bin_search
search 50  index 25
Low_bin_search
search 48  index 24
Low_bin_search
search 46  index 23
Low_bin_search
search 44  index 22
Low_bin_search
search 42  index 21
Low_bin_search
search 40  index 20
Low_bin_search
search 38  index 19
Low_bin_search
```

```
search 36  index 18
Low_bin_search
search 34  index 17
Low_bin_search
search 32  index 16
Low_bin_search
search 30  index 15
Low_bin_search
search 28  index 14
Low_bin_search
search 26  index 13
Low_bin_search
search 24  index 12
Low_bin_search
search 22  index 11
Low_bin_search
search 20  index 10
Low_bin_search
search 18  index 9
Low_bin_search
search 16  index 8
Low_bin_search
search 14  index 7
Low_bin_search
search 12  index 6
Low_bin_search
search 10  index 5
Low_bin_search
search 8  index 4
Low_bin_search
search 6  index 3
Low_bin_search
search 4  index 2
Low_bin_search
search 2  index 1
Test low_bin_nb_arithmetic
data[0] = 0
data[1] = 2
data[2] = 4
data[3] = 6
data[4] = 8
data[5] = 10
data[6] = 12
data[7] = 14
data[8] = 16
data[9] = 18
data[10] = 20
data[11] = 22
data[12] = 24
```

```
data[13] = 26
data[14] = 28
data[15] = 30
data[16] = 32
data[17] = 34
data[18] = 36
data[19] = 38
data[20] = 40
data[21] = 42
data[22] = 44
data[23] = 46
data[24] = 48
data[25] = 50
data[26] = 52
data[27] = 54
data[28] = 56
data[29] = 58
data[30] = 60
data[31] = 62
data[32] = 64
data[33] = 66
data[34] = 68
data[35] = 70
data[36] = 72
data[37] = 74
data[38] = 76
data[39] = 78
data[40] = 80
data[41] = 82
data[42] = 84
data[43] = 86
data[44] = 88
data[45] = 90
data[46] = 92
data[47] = 94
data[48] = 96
data[49] = 98
data[50] = 100
data[51] = 102
data[52] = 104
data[53] = 106
data[54] = 108
data[55] = 110
data[56] = 112
data[57] = 114
data[58] = 116
data[59] = 118
data[60] = 120
data[61] = 122
```

```
data[62] = 124
data[63] = 126
data[64] = 128
data[65] = 130
data[66] = 132
data[67] = 134
data[68] = 136
data[69] = 138
data[70] = 140
data[71] = 142
data[72] = 144
data[73] = 146
data[74] = 148
data[75] = 150
data[76] = 152
data[77] = 154
data[78] = 156
data[79] = 158
data[80] = 160
data[81] = 162
data[82] = 164
data[83] = 166
data[84] = 168
data[85] = 170
data[86] = 172
data[87] = 174
data[88] = 176
data[89] = 178
data[90] = 180
data[91] = 182
data[92] = 184
data[93] = 186
data[94] = 188
data[95] = 190
data[96] = 192
data[97] = 194
data[98] = 196
data[99] = 198
search 80  index 40
search 78  index 39
search 76  index 38
search 74  index 37
search 72  index 36
search 70  index 35
search 68  index 34
search 66  index 33
search 64  index 32
search 62  index 31
search 60  index 30
search 58  index 29
search 56  index 28
search 54  index 27
search 52  index 26
search 50  index 25
search 48  index 24
search 46  index 23
search 44  index 22
search 42  index 21
search 40  index 20
search 38  index 19
search 36  index 18
search 34  index 17
search 32  index 16
search 30  index 15
search 28  index 14
search 26  index 13
search 24  index 12
search 22  index 11
search 20  index 10
search 18  index 9
search 16  index 8
search 14  index 7
search 12  index 6
search 10  index 5
search 8  index 4
search 6  index 3
search 4  index 2
search 2  index 1
Test low_bin_mask
data[0] = 0
data[1] = 2
data[2] = 4
data[3] = 6
data[4] = 8
data[5] = 10
data[6] = 12
data[7] = 14
data[8] = 16
data[9] = 18
data[10] = 20
data[11] = 22
data[12] = 24
data[13] = 26
data[14] = 28
data[15] = 30
data[16] = 32
data[17] = 34
data[18] = 36
```

```
data[19] = 38
data[20] = 40
data[21] = 42
data[22] = 44
data[23] = 46
data[24] = 48
data[25] = 50
data[26] = 52
data[27] = 54
data[28] = 56
data[29] = 58
data[30] = 60
data[31] = 62
data[32] = 64
data[33] = 66
data[34] = 68
data[35] = 70
data[36] = 72
data[37] = 74
data[38] = 76
data[39] = 78
data[40] = 80
data[41] = 82
data[42] = 84
data[43] = 86
data[44] = 88
data[45] = 90
data[46] = 92
data[47] = 94
data[48] = 96
data[49] = 98
data[50] = 100
data[51] = 102
data[52] = 104
data[53] = 106
data[54] = 108
data[55] = 110
data[56] = 112
data[57] = 114
data[58] = 116
data[59] = 118
data[60] = 120
data[61] = 122
data[62] = 124
data[63] = 126
data[64] = 128
data[65] = 130
data[66] = 132
data[67] = 134
```

```
data[68] = 136
data[69] = 138
data[70] = 140
data[71] = 142
data[72] = 144
data[73] = 146
data[74] = 148
data[75] = 150
data[76] = 152
data[77] = 154
data[78] = 156
data[79] = 158
data[80] = 160
data[81] = 162
data[82] = 164
data[83] = 166
data[84] = 168
data[85] = 170
data[86] = 172
data[87] = 174
data[88] = 176
data[89] = 178
data[90] = 180
data[91] = 182
data[92] = 184
data[93] = 186
data[94] = 188
data[95] = 190
data[96] = 192
data[97] = 194
data[98] = 196
data[99] = 198
search 80  index 40
search 78  index 39
search 76  index 38
search 74  index 37
search 72  index 36
search 70  index 35
search 68  index 34
search 66  index 33
search 64  index 32
search 62  index 31
search 60  index 30
search 58  index 29
search 56  index 28
search 54  index 27
search 52  index 26
search 50  index 25
search 48  index 24
search 46  index 23
search 44  index 22
search 42  index 21
search 40  index 20
search 38  index 19
search 36  index 18
search 34  index 17
search 32  index 16
search 30  index 15
search 28  index 14
search 26  index 13
search 24  index 12
search 22  index 11
search 20  index 10
search 18  index 9
search 16  index 8
search 14  index 7
search 12  index 6
search 10  index 5
search 8  index 4
search 6  index 3
search 4  index 2
search 2  index 1
Test low_bin_nb_4x
data[0] = 0
data[1] = 2
data[2] = 4
data[3] = 6
data[4] = 8
data[5] = 10
data[6] = 12
data[7] = 14
data[8] = 16
data[9] = 18
data[10] = 20
data[11] = 22
data[12] = 24
data[13] = 26
data[14] = 28
data[15] = 30
data[16] = 32
data[17] = 34
data[18] = 36
data[19] = 38
data[20] = 40
data[21] = 42
data[22] = 44
data[23] = 46
data[24] = 48
```

```
data[25] = 50
data[26] = 52
data[27] = 54
data[28] = 56
data[29] = 58
data[30] = 60
data[31] = 62
data[32] = 64
data[33] = 66
data[34] = 68
data[35] = 70
data[36] = 72
data[37] = 74
data[38] = 76
data[39] = 78
data[40] = 80
data[41] = 82
data[42] = 84
data[43] = 86
data[44] = 88
data[45] = 90
data[46] = 92
data[47] = 94
data[48] = 96
data[49] = 98
data[50] = 100
data[51] = 102
data[52] = 104
data[53] = 106
data[54] = 108
data[55] = 110
data[56] = 112
data[57] = 114
data[58] = 116
data[59] = 118
data[60] = 120
data[61] = 122
data[62] = 124
data[63] = 126
data[64] = 128
data[65] = 130
data[66] = 132
data[67] = 134
data[68] = 136
data[69] = 138
data[70] = 140
data[71] = 142
data[72] = 144
data[73] = 146
```

```
data[74] = 148
data[75] = 150
data[76] = 152
data[77] = 154
data[78] = 156
data[79] = 158
data[80] = 160
data[81] = 162
data[82] = 164
data[83] = 166
data[84] = 168
data[85] = 170
data[86] = 172
data[87] = 174
data[88] = 176
data[89] = 178
data[90] = 180
data[91] = 182
data[92] = 184
data[93] = 186
data[94] = 188
data[95] = 190
data[96] = 192
data[97] = 194
data[98] = 196
data[99] = 198
search 80  index 40
search 78  index 39
search 76  index 38
search 74  index 37
search 72  index 36
search 70  index 35
search 68  index 34
search 66  index 33
search 64  index 32
search 62  index 31
search 60  index 30
search 58  index 29
search 56  index 28
search 54  index 27
search 52  index 26
search 50  index 25
search 48  index 24
search 46  index 23
search 44  index 22
search 42  index 21
search 40  index 20
search 38  index 19
search 36  index 18
search 34  index 17
search 32  index 16
search 30  index 15
search 28  index 14
search 26  index 13
search 24  index 12
search 22  index 11
search 20  index 10
search 18  index 9
search 16  index 8
search 14  index 7
search 12  index 6
search 10  index 5
search 8  index 4
search 6  index 3
search 4  index 2
search 2  index 1
db5242test.c: In function ‘main’:
db5242test.c:663:43: error: expected ‘;’ before ‘int64_t’
  663 |     printf("Test low_bin_nb_simd simd \n")
      |                                           ^
      |                                           ;
  664 |     int64_t arraysize;
      |     ~~~~~~~
Test low_bin_nb_4x
data[0] = 0
data[1] = 2
data[2] = 4
data[3] = 6
data[4] = 8
data[5] = 10
data[6] = 12
data[7] = 14
data[8] = 16
data[9] = 18
data[10] = 20
data[11] = 22
data[12] = 24
data[13] = 26
data[14] = 28
data[15] = 30
data[16] = 32
data[17] = 34
data[18] = 36
data[19] = 38
data[20] = 40
data[21] = 42
data[22] = 44
data[23] = 46
```

```
data[24] = 48
data[25] = 50
data[26] = 52
data[27] = 54
data[28] = 56
data[29] = 58
data[30] = 60
data[31] = 62
data[32] = 64
data[33] = 66
data[34] = 68
data[35] = 70
data[36] = 72
data[37] = 74
data[38] = 76
data[39] = 78
data[40] = 80
data[41] = 82
data[42] = 84
data[43] = 86
data[44] = 88
data[45] = 90
data[46] = 92
data[47] = 94
data[48] = 96
data[49] = 98
data[50] = 100
data[51] = 102
data[52] = 104
data[53] = 106
data[54] = 108
data[55] = 110
data[56] = 112
data[57] = 114
data[58] = 116
data[59] = 118
data[60] = 120
data[61] = 122
data[62] = 124
data[63] = 126
data[64] = 128
data[65] = 130
data[66] = 132
data[67] = 134
data[68] = 136
data[69] = 138
data[70] = 140
data[71] = 142
data[72] = 144
```

```
data[73] = 146
data[74] = 148
data[75] = 150
data[76] = 152
data[77] = 154
data[78] = 156
data[79] = 158
data[80] = 160
data[81] = 162
data[82] = 164
data[83] = 166
data[84] = 168
data[85] = 170
data[86] = 172
data[87] = 174
data[88] = 176
data[89] = 178
data[90] = 180
data[91] = 182
data[92] = 184
data[93] = 186
data[94] = 188
data[95] = 190
data[96] = 192
data[97] = 194
data[98] = 196
data[99] = 198
search 80  index 40
search 78  index 39
search 76  index 38
search 74  index 37
search 72  index 36
search 70  index 35
search 68  index 34
search 66  index 33
search 64  index 32
search 62  index 31
search 60  index 30
search 58  index 29
search 56  index 28
search 54  index 27
search 52  index 26
search 50  index 25
search 48  index 24
search 46  index 23
search 44  index 22
search 42  index 21
search 40  index 20
search 38  index 19
search 36  index 18
search 34  index 17
search 32  index 16
search 30  index 15
search 28  index 14
search 26  index 13
search 24  index 12
search 22  index 11
search 20  index 10
search 18  index 9
search 16  index 8
search 14  index 7
search 12  index 6
search 10  index 5
search 8  index 4
search 6  index 3
search 4  index 2
search 2  index 1
```

Test the band_join (0/2)

```
Test Failed: 28416 != 0
```

Test the band_join with SIMD (0/2)

```
Test Failed: 28416 != 0
```

Test the low bin search and it should always pass (2/2)

Compilation with mavx2 (2/2)

Compilation with mavx512f (2/2)

Test the low_bin_nb_arithmetic (2/2)

Test the low_bin_nb_mask (2/2)

Test the low_bin_nb_4x (2/2)

Test the low_bin_nb_simd (2/2)

Submitted Files

▾ db5242.c

Download

```
/*
  CSE 5242 Project 2, Fall 2023

  See class project handout for more extensive documentation.

  https://stackoverflow.com/questions/19068705/undefined-reference-when-calling-inline-function
*/

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <asm/unistd.h>
#include <immintrin.h>

/* uncomment out the following DEBUG line for debug info, for experiment comment the DEBUG line */
//#define DEBUG


/* compare two int64_t values - for use with qsort */
static int compare(const void *p1, const void *p2)
{
  int a,b;
  a = *(int64_t *)p1;
  b = *(int64_t *)p2;
  if (a<b) return -1;
  if (a==b) return 0;
  return 1;
}

/* initialize searches and data - data is sorted and searches is a random permutation of data */
int init(int64_t* data, int64_t* searches, int count)
{
  for(int64_t i=0; i<count; i++){
    searches[i] = random();
    data[i] = searches[i]+1;
  }
  qsort(data,count,sizeof(int64_t),compare);
}

/* initialize outer probes of band join */
int band_init(int64_t* outer, int64_t size)
```

```
{
  for(int64_t i=0; i<size; i++){
    outer[i] = random();
  }
}

inline int64_t simple_binary_search(int64_t* data, int64_t size, int64_t target)
{
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  while(left<=right) {
    mid = (left + right)/2;   /* ignore possibility of overflow of left+right */
    if (data[mid]==target) return mid;
    if (data[mid]<target) left=mid+1;
    else right = mid-1;
  }
  return -1; /* no match */
}

inline int64_t low_bin_search(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
     (a) does only one comparison in the inner loop
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  printf("Low_bin_search\n");
  while(left<right) {
    mid = (left + right)/2;   /* ignore possibility of overflow of left+right */
    if (data[mid]>=target)
      right=mid;
    else
      left=mid+1;
  }
  return right;
}

//#define ARDEBUG
inline int64_t low_bin_nb_arithmetic(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
```

```
     (a) does no comparisons in the inner loop by using multiplication and addition to convert control dependencies
         to data dependencies
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  #ifdef ARDEBUG
    printf("low_bin_nb_arith\n");
  #endif
  //0^1 = 1 and 1^1 = 0
  while(left<right) {
    mid = (left + right) / 2; //get middle
    int64_t flag = data[mid] >= target; //data[mid] >= target ? 1 : 0 (if target value is left side of the middle index flag = 1 else 0)
    left = flag * left + (flag^1) * (mid+1);// if flag == 1 assign previous left value (left index stays same) else assign mid+1 value to divide the array into half
    right = flag * mid + (flag^1) * right;// if flag == 1 assign mid value to divide array into half else assign previous right value (right index stays same)
    #ifdef ARDEBUG
        printf("mid: %d left: %d right: %d\n", mid, left, right);
    #endif
  }
  return right;
}

//#define MASTDEBUG
inline int64_t low_bin_nb_mask(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations to convert control dependencies
         to data dependencies
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  #ifdef MASTDEBUG
    printf("low_bin_nb_mask\n");
  #endif
  // ~(-1) = 0  and ~(0) = -1
  //-1 & 3 = 3  and 0 & 3 = 0
  //0 | 3 = 3
  while(left<right) {
     //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
    mid = (left + right) / 2;   //get middle
    int64_t flag = (data[mid] >= target)-1;  //data[mid] >= target ? 0 : -1 (if target value is left side of middle index flag = 0 else =1)
    left = (flag & (mid+1)) | (~flag & left); // If flag == -1 assign mid+1 to divide the array into half else assign previous left value (left index stays same)
    right = (flag & right) | (~flag & mid); // if flag == -1 assign previous right value (right index stay same) else assing mid to divide the array into half
    #ifdef MASTDEBUG
        printf("mid: %d left: %d right: %d\n", mid, left, right);
    #endif
    //int64_t flag = (data[mid] >= target);  //data[mid] >= target ? 1 : 0
    // left = ((flag-1) & (mid+1)) | ((flag^1)-1 & left);
    // right = ((flag-1) & right) | ((flag^1)-1 & mid);
  }
  return right;
}

//#define XDEBUG
inline void low_bin_nb_4x(int64_t* data, int64_t size, int64_t* targets, int64_t* right)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations instead
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
     (d) does 4 searches at the same time in an interleaved fashion, so that an out-of-order processor
         can make progress even when some instructions are still waiting for their operands to be ready.

     Note that we're using the result array as the "right" elements in the search so no need for a return statement
  */
  int64_t left[4]={0,0,0,0};
  int64_t mid[4];
  right[0]=right[1]=right[2]=right[3]=size;

  #ifdef XDEBUG
    printf("low_bin_nb_4x\n");
  #endif
  int64_t bit_size = 4;
  int cnt = 0;
  //outer loop to check termination condition like above (e.g left < right)
  while (left[0]<right[0] || left[1]<right[1] || left[2]<right[2] || left[3]<right[3]) {
    int i;
    #ifdef XDEBUG
        printf("cnt: %d\n", cnt);
    #endif
    //inner loop to perform 4 search concurrently
    for (i = 0; i < bit_size; i++) {
      //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
      mid[i] = (left[i] + right[i]) / 2;   //get middle
      int64_t flag = (data[mid[i]] >= targets[i])-1;  //data[mid] >= target ? 0 : -1  (if target value is left side of middle index flag = 0 else =1)
      left[i] = (flag & (mid[i]+1)) | (~flag & left[i]);  // If flag == -1 assign mid+1 to divide the array into half else assign previous left value (left index stays same)
      right[i] = (flag & right[i]) | (~flag & mid[i]); // if flag == -1 assign previous right value (right index stay same) else assing mid to divide the array into half
      #ifdef XDEBUG
            printf("mid: %d left: %d right: %d\n", mid[i], left[i], right[i]);
      #endif

      //int64_t flag = (data[mid[i]] > targets[i]);  //data[mid] >= target ? 1 : 0
      // left[i] = ((flag-1) & (mid[i]+1)) | ((flag^1)-1 & left[i]);
      // right[i] = ((flag-1) & right[i]) | ((flag^1)-1 & mid[i]);
      //break;
    }
    cnt+=1;
  }
}


/* The following union type is handy to output the contents of AVX512 data types */
union int8x4 {
  __m256i a;
  int64_t b[4];
};

void printavx(char* name, __m256i v) {
  union int8x4 n;

  n.a=v;
  printf("Value in %s is [%ld %ld %ld %ld ]\n",name,n.b[0],n.b[1],n.b[2],n.b[3]);
}

/*
 * Optinal for using AVX-512

  union int8x8 {
    __m512i a;
    int64_t b[8];
  };

  void printavx512(char* name, __m512i v) {
    union int8x4 n;

    n.a=v;
    printf("Value in %s is [%ld %ld %ld %ld %ld %ld %ld %ld ]\n",name,n.b[0],n.b[1],n.b[2],n.b[3]);
  }

 */

//#define SIMDDEBUG
inline void low_bin_nb_simd(int64_t* data, int64_t size, __m256i target, __m256i* result)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations instead
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
     (d) does 4 searches at the same time using AVX2 intrinsics

     See https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-8/intrinsics-for-avx2.html
     for documentation of the AVX512 intrinsics

     Note that we're using the result array as the "right" elements in the search, and that searchkey is being passed
     as an __m256i value rather than via a pointer.
  */

 __m256i aleft = _mm256_set1_epi64x(0);
  __m256i aright = _mm256_set1_epi64x(size);
  __m256i amid;

  __m256i ones = _mm256_set1_epi64x(1);
  __m256i amask;
  __m256i datavec;

#ifdef SIMDDEBUG
  printf("low_bin_nb_simd\n");
#endif
  //AVX Intrinsic Guide: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
  __m256i cmp = _mm256_cmpgt_epi64(aright, aleft);
  //Check if all elements of vector is 0 using testz: https://stackoverflow.com/questions/32072169/could-i-compare-to-zero-register-in-avx-correctly + https://stackoverflow.com/a/32120039
  //Check all elements in cmp are 0 (if value of cmp is 0 it means left > right else -1)
  while(_mm256_testz_si256(cmp, cmp) == 0){
    amid = _mm256_srli_epi64(_mm256_add_epi64(aleft, aright), 1); //get middle
    __m256i amid_one = _mm256_add_epi64(amid, ones); //get middle+1 vector
    //Extract matching values of data using AVX index vector using i64gather: https://stackoverflow.com/questions/51128005/what-do-you-do-without-fast-gather-and-scatter-in-avx2-instructions
    datavec = _mm256_i64gather_epi64((long long*)data, amid, sizeof(int64_t));

    // inlining failed error for >= opeartion
    //__mmask8 flag = _mm256_cmpge_epi64_mask(datavec, target);
    //Combine > operation and = operation to have >= operation: https://www.splashlearn.com/math-vocabulary/greater-than-or-equal-to
    __m256i cmpgt = _mm256_cmpgt_epi64(datavec, target);
    __m256i cmpeq = _mm256_cmpeq_epi64(datavec, target);
    __m256i flag = _mm256_or_si256(cmpgt, cmpeq);

    //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
    aleft = _mm256_or_si256(_mm256_and_si256(flag, aleft),_mm256_andnot_si256(flag, amid_one)); // if falg == -1 assign left value (left index stays same) else assing mid+1 to divide the array into half.
    aright = _mm256_or_si256(_mm256_and_si256(flag, amid),_mm256_andnot_si256(flag, aright)); // if flag == -1 assign mid value to divide array into half else assign previous right value (rihgt index stays same)

#ifdef SIMDDEBUG
    printavx("amid", amid);
    printavx("flag", flag);
    printavx("aright", aright);
    printavx("aleft", aleft);
    printf("\n\n");
#endif
    //break;
    cmp = _mm256_cmpgt_epi64(aright, aleft);
  }
  //aright = _mm256_sub_epi64(aright, ones);
  *result = aright;
}

void bulk_bin_search(int64_t* data, int64_t size, int64_t* searchkeys, int64_t numsearches, int64_t* results, int repeats)
{
  for(int j=0; j<repeats; j++) {
    /* Function to test a large number of binary searches

       we might need repeats>1 to make sure the events we're measuring are not dominated by various
       overheads, particularly for small values of size and/or numsearches

       we assume that we want exactly "size" searches, where "size" is the length if the searchkeys array
     */
    for(int64_t i=0;i<numsearches; i++) {
#ifdef DEBUG
```

```
      printf("Searching for %ld...\n",searchkeys[i]);
#endif

      // Uncomment one of the following to measure it
      //results[i] = low_bin_search(data,size,searchkeys[i]);
      //results[i] = low_bin_nb_arithmetic(data,size,searchkeys[i]);
      results[i] = low_bin_nb_mask(data,size,searchkeys[i]);

#ifdef DEBUG
      printf("Result is %ld\n",results[i]);
#endif
    }
  }
}

void bulk_bin_search_4x(int64_t* data, int64_t size, int64_t* searchkeys, int64_t numsearches, int64_t* results, int repeats)
{
  register __m256i searchkey_4x;

  for(int j=0; j<repeats; j++) {
    /* Function to test a large number of binary searches using one of the 8x routines

       we might need repeats>1 to make sure the events we're measuring are not dominated by various
       overheads, particularly for small values of size and/or numsearches

       we assume that we want exactly "size" searches, where "size" is the length if the searchkeys array
     */
    int64_t extras = numsearches % 4;
    for(int64_t i=0;i<numsearches-extras; i+=4) {
#ifdef DEBUG
      printf("Searching for %ld %ld %ld %ld  ...\n",
             searchkeys[i],searchkeys[i+1],searchkeys[i+2],searchkeys[i+3]);
#endif

      // Uncomment one of the following depending on which routine you want to profile

      // Algorithm A
      //low_bin_nb_4x(data,size,&searchkeys[i],&results[i]);

      // Algorithm B
      searchkey_4x = _mm256_loadu_si256((__m256i *)&searchkeys[i]);
      low_bin_nb_simd(data,size,searchkey_4x,(__m256i *)&results[i]);

#ifdef DEBUG
      printf("Result is %ld %ld %ld %ld  ...\n",
            results[i],results[i+1],results[i+2],results[i+3]);
#endif
    }
    /* a little bit more work if numsearches is not a multiple of 8 */
    for(int64_t i=numsearches-extras;i<numsearches; i++) {

      results[i] = low_bin_nb_mask(data,size,searchkeys[i]);

    }

  }
}


int64_t band_join(int64_t* inner, int64_t inner_size, int64_t* outer, int64_t outer_size, int64_t* inner_results, int64_t* outer_results, int64_t result_size, int64_t bound)
{
  /* In a band join we want matches within a range of values.  If p is the probe value from the outer table, then all
     reccords in the inner table with a key in the range [p-bound,p+bound] inclusive should be part of the result.

     Results are returned via two arrays. outer_results stores the index of the outer table row that matches, and
     inner_results stores the index of the inner table row that matches.  result_size tells you the size of the
     output array that has been allocated. You should make sure that you don't exceed this size.  If there are
     more results than can fit in the result arrays, then return early with just a prefix of the results in the result
     arrays. The return value of the function should be the number of output results.

  */
 //Declaring arrays to store left and right index of the range.
  int64_t *leftIndexArray = malloc(outer_size * sizeof(int64_t));
  int64_t *rightIndexArray = malloc(outer_size * sizeof(int64_t));
  if (!leftIndexArray || !rightIndexArray) {
    free(leftIndexArray);
    free(rightIndexArray);
    // Memory allocation failed, so we return -1
    return -1;
  }
  //Calcuating below values to find the multiplier and remainder.
  int multiplier = outer_size/4;
  int remainder = outer_size%4;

  int temp_1=0;

  while (temp_1<multiplier){
    // printf("Inside  muliplier with i->%d\n",temp_1);
    //using the below arrays to store the lower bound values temporarily after using low_bin_nb_mask function.
    int64_t lower_limit[4];
    int64_t upper_limit[4];

    for (int i =0;i<4;i++){
      lower_limit[i]=outer[4*temp_1+i]-bound;
      upper_limit[i]=outer[4*temp_1+i]+bound;
    }
    int64_t left_temp_Array[4];
    int64_t right_temp_Array[4];

    low_bin_nb_4x(inner,inner_size,lower_limit,left_temp_Array);
    low_bin_nb_4x(inner,inner_size,upper_limit,right_temp_Array);

    for (int i =0;i<4;i++){
      leftIndexArray[4*temp_1 + i]=left_temp_Array[i];
      rightIndexArray[4*temp_1 + i]=right_temp_Array[i];
    }
    // printf("Ending  muliplier with i->%d\n",temp_1);
    temp_1+=1;

  }

  for(int i = 0; i<remainder;i++){
    // printf("Entering remainder with i->%d\n",i);
    // printf("lower-> %d\n",outer[i]-bound);
    // printf("upper-> %d\n",outer[i]+bound);
    leftIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner, inner_size, outer[4*temp_1 + i]-bound);//array containing left index in the range.
    rightIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner,inner_size,outer[4*temp_1 + i]+bound);//array containing right index of the range.
    // printf("leftindex->%d \t\n rightindex->%d\t\n",leftIndexArray[i],rightIndexArray[i]);
    // printf("Exiting reminder loop with i->%d\n",i);
  }printf("\n");



/* ONLY WITH low_bin_nb_mask
  for(int i = 0; i<outer_size;i++){
    // printf("lower-> %d\n",outer[i]-bound);
    // printf("upper-> %d\n",outer[i]+bound);
    leftIndexArray[i]=low_bin_nb_mask(inner, inner_size, outer[i]-bound);//array containing left index in the range.
    rightIndexArray[i]=low_bin_nb_mask(inner,inner_size,outer[i]+bound);//array containing right index of the range.
    // printf("leftindex->%d \t\n rightindex->%d\t\n",leftIndexArray[i],rightIndexArray[i]);
  }printf("\n");
*/
// for(int i = 0; i<outer_size;i++){
// printf("leftIndexArray %d->%d and rightIndexArray %d -
>%d\n",i,leftIndexArray[i],i,rightIndexArray[i]);
// }

  int temp=0;

  for(int i = 0; i<outer_size;i++){
    // printf("Entering inside the merging part with i->%d\n",i);
    int left_index = leftIndexArray[i];
    int right_index = rightIndexArray[i];
    //if left index and right index are same then we skip the iteration because there are no elements.
    if (left_index==right_index){continue;}

    while ((temp<result_size) && (left_index<right_index)){
      // printf("Inside while for outer element index %d\n",i);
      outer_results[temp]=i;
      inner_results[temp]=left_index;
      // printf("adding values (%d,%d) in outer and inner result\n",outer_results[i],inner_results[i]);
      left_index+=1;
      temp+=1;

    }
//break if we reach the result size.
    if (temp==result_size){break;}
    // printf("Exiting merging part with i->%d\n",i);

  }

  // for (int i = 0; i<temp;i++){
  //   printf("(OuterResult,InnerResult)->(%ld,%ld)\n",outer_results[i],inner_results[i]);
  // }

  free(leftIndexArray);
  free(rightIndexArray);

  return temp;

}

int64_t band_join_simd(int64_t* inner, int64_t inner_size, int64_t* outer, int64_t outer_size, int64_t* inner_results, int64_t* outer_results, int64_t result_size, int64_t bound)
{
  /* In a band join we want matches within a range of values.  If p is the probe value from the outer table, then all
     reccords in the inner table with a key in the range [p-bound,p+bound] inclusive should be part of the result.

     Results are returned via two arrays. outer_results stores the index of the outer table row that matches, and
     inner_results stores the index of the inner table row that matches.  result_size tells you the size of
```

```
the
    output array that has been allocated. You should make sure that you don't exceed this size.  If there are
    more results than can fit in the result arrays, then return early with just a prefix of the results in the result
    arrays. The return value of the function should be the number of output results.

    To do the binary search, you could use the low_bin_nb_simd you just implemented to search for the lower bounds in parallel

    Once you've found the lower bounds, do the following for each of the 4 search keys in turn:
       scan along the sorted inner array, generating outputs for each match, and making sure not to exceed the output array bounds.

    This inner scanning code does not have to use SIMD.
  */



 //Using similiar approach from the above function.

  // Create and allocate space to store left and right result indexes
  int64_t *leftIndexArray = malloc(outer_size * sizeof(int64_t));
  int64_t *rightIndexArray = malloc(outer_size * sizeof(int64_t));
  if (!leftIndexArray || !rightIndexArray) {
    free(leftIndexArray);
    free(rightIndexArray);
    return -1;
  }

  // Calculate multiplier and remainder to find number of simd function instances and individual mask function call instances
  int multiplier = outer_size/4;
  int remainder = outer_size%4;

  int temp_1=0;
  while (temp_1<multiplier){
    // compute lower and upper bound vectors using AVX functions
    __m256i outer_vec = _mm256_loadu_si256((__m256i*)&outer[4*temp_1]);
    __m256i lower_bound_vec = _mm256_sub_epi64(outer_vec, _mm256_set1_epi64x(bound));
    __m256i upper_bound_vec = _mm256_add_epi64(outer_vec, _mm256_set1_epi64x(bound));

    // Create temporary indexes to store each iteration's results
    int64_t left_indexes[4];
    int64_t right_indexes[4];

    // Call binary search function using simd
    low_bin_nb_simd(inner,inner_size,lower_bound_vec,(__m256i*) left_indexes);
    low_bin_nb_simd(inner,inner_size,upper_bound_vec,(__m256i*) right_indexes);

    // Store results in temporary arrays
    for (int i =0;i<4;i++){
      leftIndexArray[4*temp_1 + i]=left_indexes[i];
      rightIndexArray[4*temp_1 + i]=right_indexes[i];
    }
    temp_1+=1;

  }

  // Store indexes of remaining cases into the result arrays
  for(int i = 0; i<remainder;i++){
    leftIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner, inner_size, outer[4*temp_1 + i]-bound);//array containing left index in the range.
    rightIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner,inner_size,outer[4*temp_1 + i]+bound);//array containing right index of the range.
  }

  // temp will be the final result size
  int temp=0;

  for(int i = 0; i<outer_size;i++){
    int left_index = leftIndexArray[i];
    int right_index = rightIndexArray[i];

    if (left_index==right_index){continue;}

    // Store indexes into the result format
    while ((temp<result_size) && (left_index<right_index)){
      outer_results[temp]=i;
      inner_results[temp]=left_index;
      left_index+=1;
      temp+=1;

    }

    if (temp==result_size){break;}

  }

  // Clear index array space
  free(leftIndexArray);
  free(rightIndexArray);

  // Reuturn result size
  return temp;


 /*
```

```
  INITIAL CODE WHICH WAS GIVING INCONSISTENT RESULT. PLEASE SKIP TO END OF COMMENT FOR THE FINAL CODE.
 int size = 0;

  for(int i=0; (i<outer_size) && (size<result_size); i+=4){
    __m256i outer_vec = _mm256_loadu_si256((__m256i*)&outer[i]);
    __m256i lower_bound_vec = _mm256_sub_epi64(outer_vec, _mm256_set1_epi64x(bound));
    __m256i upper_bound_vec = _mm256_add_epi64(outer_vec, _mm256_set1_epi64x(bound));

    int64_t left_indexes[4];
    int64_t right_indexes[4];

    int remainder = outer_size - i;
    if (remainder >= 4) {

      __m256i left_result[1];
      __m256i right_result[1];

      low_bin_nb_simd(inner,inner_size,lower_bound_vec,left_result);
      low_bin_nb_simd(inner,inner_size,upper_bound_vec,right_result);

      _mm256_store_si256((__m256i*)left_indexes, left_result[0]);
      _mm256_store_si256((__m256i*)right_indexes, right_result[0]);

      for(int j = 0; j < 4; j++) {
        while((size < result_size) && (left_indexes[j] < right_indexes[j])) {
          outer_results[size] = i+j;
          inner_results[size] = left_indexes[j];
          left_indexes[j]++;
          size++;
        }
      }

    } else {

      if (remainder >= 1) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 0);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 0);

        left_indexes[0] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[0] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      if (remainder >= 2) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 1);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 1);

        left_indexes[1] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[1] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      if (remainder == 3) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 2);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 2);

        left_indexes[2] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[2] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      for(int j = 0; j < remainder; j++) {
        while((size < result_size) && (left_indexes[j] < right_indexes[j])) {
          outer_results[size] = i+j;
          inner_results[size] = left_indexes[j];
          left_indexes[j]++;
          size++;
        }
      }

    }

  }
  return size;
  */

}

int main(int argc, char *argv[])
{
         long long counter;
         int64_t arraysize, outer_size, result_size;
         int64_t bound;
         int64_t *data, *queries, *results;
         int ret;
         struct timeval before, after;
         int repeats;
         int64_t total_results;

         // band-join arrays
         int64_t *outer, *outer_results, *inner_results;


         if (argc >= 5)
           {
             arraysize = atoi(argv[1]);
             outer_size = atoi(argv[2]);
             result_size = atoi(argv[3]);
             bound = atoi(argv[4]);
           }
         else
           {
             fprintf(stderr, "Usage: db5242 inner_size outer_size result_size bound <repeats>\n");
             exit(EXIT_FAILURE);
           }

         if (argc >= 6)
           repeats = atoi(argv[5]);
         else
           {
             repeats=1;
           }
     //  printf("InsideMain and bound->%d\n",bound);

         /* allocate the array and the queries for searching */
         ret=posix_memalign((void**) &data,64,arraysize*sizeof(int64_t));
         if (ret)
         {
           fprintf(stderr, "Memory allocation error.\n");
           exit(EXIT_FAILURE);
         }
         ret=posix_memalign((void**) &queries,64,arraysize*sizeof(int64_t));
         if (ret)
         {
           fprintf(stderr, "Memory allocation error.\n");
           exit(EXIT_FAILURE);
         }
         ret=posix_memalign((void**) &results,64,arraysize*sizeof(int64_t));
         if (ret)
         {
           fprintf(stderr, "Memory allocation error.\n");
           exit(EXIT_FAILURE);
         }

         /* allocate the outer array and output arrays for band-join */
         ret=posix_memalign((void**) &outer,64,outer_size*sizeof(int64_t));
         if (ret)
         {
           fprintf(stderr, "Memory allocation error.\n");
           exit(EXIT_FAILURE);
         }
         ret=posix_memalign((void**) &outer_results,64,result_size*sizeof(int64_t));
         if (ret)
         {
           fprintf(stderr, "Memory allocation error.\n");
           exit(EXIT_FAILURE);
         }
         ret=posix_memalign((void**) &inner_results,64,result_size*sizeof(int64_t));
         if (ret)
```

```
       {
        fprintf(stderr, "Memory allocation error.\n");
        exit(EXIT_FAILURE);
       }


        /* code to initialize data structures goes here so that it is not included in the timing measurement */
        init(data,queries,arraysize);
        band_init(outer,outer_size);

#ifdef DEBUG
        /* show the arrays */
        printf("data: ");
        for(int64_t i=0;i<arraysize;i++) printf("%ld ",data[i]);
        printf("\n");
        printf("queries: ");
        for(int64_t i=0;i<arraysize;i++) printf("%ld ",queries[i]);
        printf("\n");
        printf("outer: ");
        for(int64_t i=0;i<outer_size;i++) printf("%ld ",outer[i]);
        printf("\n");
#endif


        /* now measure... */

        gettimeofday(&before,NULL);

        /* the code that you want to measure goes here; make a function call */
    printf("bulk_bin_search_start\n");
        bulk_bin_search(data,arraysize,queries,arraysize,results, repeats);

        gettimeofday(&after,NULL);
        printf("Time in bulk_bin_search loop is %ld microseconds or %f microseconds per search\n", (after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec), 1.0*((after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec))/arraysize/repeats);



        gettimeofday(&before,NULL);

        /* the code that you want to measure goes here; make a function call */
    printf("bulk_bin_search_4x_start\n");
        bulk_bin_search_4x(data,arraysize,queries,arraysize,results, repeats);

        gettimeofday(&after,NULL);
        printf("Time in bulk_bin_search_4x loop is %ld microseconds or %f microseconds per search\n", (after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec), 1.0*((after.tv_sec-
before.tv_sec)*1000000+(after.tv_usec-before.tv_usec))/arraysize/repeats);



        gettimeofday(&before,NULL);

        /* the code that you want to measure goes here; make a function call */
        total_results=band_join(data, arraysize, outer, outer_size, inner_results, outer_results, result_size, bound);

        gettimeofday(&after,NULL);
        printf("Band join result size is %ld with an average of %f matches per output record\n",total_results, 1.0*total_results/(1.0+outer_results[total_results-1]));
        printf("Time in band_join loop is %ld microseconds or %f microseconds per outer record\n", (after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec), 1.0*((after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec))/outer_size);

#ifdef DEBUG
        /* show the band_join results */
        printf("band_join results: ");
        for(int64_t i=0;i<total_results;i++) printf("(%ld,%ld) ",outer_results[i],inner_results[i]);
        printf("\n");

#endif


    gettimeofday(&before,NULL);

        /* the code that you want to measure goes here; make a function call */
        total_results=band_join_simd(data, arraysize, outer, outer_size, inner_results, outer_results, result_size, bound);

        gettimeofday(&after,NULL);
        printf("Band join (SIMD) result size is %ld with an average of %f matches per output record\n",total_results, 1.0*total_results/(1.0+outer_results[total_results-1]));
        printf("Time in band_join_simd loop is %ld microseconds or %f microseconds per outer record\n", (after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec), 1.0*((after.tv_sec-before.tv_sec)*1000000+(after.tv_usec-before.tv_usec))/outer_size);

#ifdef DEBUG
        /* show the band_join results */
        printf("band_join_simd results: ");
        for(int64_t i=0;i<total_results;i++) printf("(%ld,%ld) ",outer_results[i],inner_results[i]);
        printf("\n");
#endif


    // FILE *csvFile;
    // csvFile = fopen("band_join.csv", "w");
    // fprintf(csvFile, "inner, outer\n");
        // for(int64_t i=0;i<total_results1;i++){
    //   fprintf(csvFile, "%ld, %ld\n", inner_results[i],outer_results[i]);
    // }
        // fclose(csvFile);

        // FILE *csvFile2;
    // csvFile2 = fopen("band_join_SIMD.csv", "w");
    // fprintf(csvFile2, "inner, outer\n");
        // for(int64_t i=0;i<total_results2;i++){
    //   fprintf(csvFile2, "%ld, %ld\n", inner_results[i],outer_results[i]);
    // }
        // fclose(csvFile2);

    // FILE *csvFile3;
    // csvFile3 = fopen("data.csv", "w");
    // fprintf(csvFile3, "inner, outer\n");
    // for(int64_t i=0;i<arraysize;i++){
    //   fprintf(csvFile3, "%ld, %ld\n", data[i], outer[i]);
    // }
    // fclose(csvFile3);


}


```

▾ db5242.inp Download

```
/*
  CSE 5242 Project 2, Fall 2023

  See class project handout for more extensive documentation.

  https://stackoverflow.com/questions/19068705/undefined-reference-when-calling-inline-function
*/

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <asm/unistd.h>
#include <immintrin.h>

/* uncomment out the following DEBUG line for debug info, for experiment comment the DEBUG line */
//#define DEBUG


/* compare two int64_t values - for use with qsort */
static int compare(const void *p1, const void *p2)
{
  int a,b;
  a = *(int64_t *)p1;
  b = *(int64_t *)p2;
  if (a<b) return -1;
  if (a==b) return 0;
  return 1;
}

/* initialize searches and data - data is sorted and searches is a random permutation of data */
int init(int64_t* data, int64_t* searches, int count)
{
  for(int64_t i=0; i<count; i++){
    searches[i] = random();
    data[i] = searches[i]+1;
  }
  qsort(data,count,sizeof(int64_t),compare);
}

/* initialize outer probes of band join */
int band_init(int64_t* outer, int64_t size)
```

```
{
  for(int64_t i=0; i<size; i++){
    outer[i] = random();
  }
}

inline int64_t simple_binary_search(int64_t* data, int64_t size, int64_t target)
{
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  while(left<=right) {
    mid = (left + right)/2;   /* ignore possibility of overflow of left+right */
    if (data[mid]==target) return mid;
    if (data[mid]<target) left=mid+1;
    else right = mid-1;
  }
  return -1; /* no match */
}

inline int64_t low_bin_search(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
     (a) does only one comparison in the inner loop
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  printf("Low_bin_search\n");
  while(left<right) {
    mid = (left + right)/2;   /* ignore possibility of overflow of left+right */
    if (data[mid]>=target)
      right=mid;
    else
      left=mid+1;
  }
  return right;
}

//#define ARDEBUG
inline int64_t low_bin_nb_arithmetic(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
```

```
     (a) does no comparisons in the inner loop by using multiplication and addition to convert control dependencies
         to data dependencies
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  #ifdef ARDEBUG
    printf("low_bin_nb_arith\n");
  #endif
  //0^1 = 1 and 1^1 = 0
  while(left<right) {
    mid = (left + right) / 2; //get middle
    int64_t flag = data[mid] >= target; //data[mid] >= target ? 1 : 0 (if target value is left side of the middle index flag = 1 else 0)
    left = flag * left + (flag^1) * (mid+1);// if flag == 1 assign previous left value (left index stays same) else assign mid+1 value to divide the array into half
    right = flag * mid + (flag^1) * right;// if flag == 1 assign mid value to divide array into half else assign previous right value (right index stays same)
    #ifdef ARDEBUG
        printf("mid: %d left: %d right: %d\n", mid, left, right);
    #endif
  }
  return right;
}

//#define MASTDEBUG
inline int64_t low_bin_nb_mask(int64_t* data, int64_t size, int64_t target)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations to convert control dependencies
         to data dependencies
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
  */
  int64_t left=0;
  int64_t right=size;
  int64_t mid;

  #ifdef MASTDEBUG
    printf("low_bin_nb_mask\n");
  #endif
  // ~(-1) = 0  and ~(0) = -1
  //-1 & 3 = 3  and 0 & 3 = 0
  //0 | 3 = 3
  while(left<right) {
     //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
    mid = (left + right) / 2;   //get middle
    int64_t flag = (data[mid] >= target)-1;  //data[mid] >= target ? 0 : -1 (if target value is left side of middle index flag = 0 else =1)
    left = (flag & (mid+1)) | (~flag & left); // If flag == -1 assign mid+1 to divide the array into half else assign previous left value (left index stays same)
    right = (flag & right) | (~flag & mid); // if flag == -1 assign previous right value (right index stay same) else assing mid to divide the array into half
    #ifdef MASTDEBUG
        printf("mid: %d left: %d right: %d\n", mid, left, right);
    #endif
    //int64_t flag = (data[mid] >= target);  //data[mid] >= target ? 1 : 0
    // left = ((flag-1) & (mid+1)) | ((flag^1)-1 & left);
    // right = ((flag-1) & right) | ((flag^1)-1 & mid);
  }
  return right;
}

//#define XDEBUG
inline void low_bin_nb_4x(int64_t* data, int64_t size, int64_t* targets, int64_t* right)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations instead
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
     (d) does 4 searches at the same time in an interleaved fashion, so that an out-of-order processor
         can make progress even when some instructions are still waiting for their operands to be ready.

     Note that we're using the result array as the "right" elements in the search so no need for a return statement
  */
  int64_t left[4]={0,0,0,0};
  int64_t mid[4];
  right[0]=right[1]=right[2]=right[3]=size;

  #ifdef XDEBUG
    printf("low_bin_nb_4x\n");
  #endif
  int64_t bit_size = 4;
  int cnt = 0;
  //outer loop to check termination condition like above (e.g left < right)
```

```c
  while (left[0]<right[0] || left[1]<right[1] || left[2]<right[2] || left[3]<right[3]) {
    int i;
    #ifdef XDEBUG
        printf("cnt: %d\n", cnt);
    #endif
    //inner loop to perform 4 search concurrently
    for (i = 0; i < bit_size; i++) {
      //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
      mid[i] = (left[i] + right[i]) / 2;   //get middle
      int64_t flag = (data[mid[i]] >= targets[i])-1;  //data[mid] >= target ? 0 : -1  (if target value is left side of middle index flag = 0 else =1)
      left[i] = (flag & (mid[i]+1)) | (~flag & left[i]);  // If flag == -1 assign mid+1 to divide the array into half else assign previous left value (left index stays same)
      right[i] = (flag & right[i]) | (~flag & mid[i]); // if flag == -1 assign previous right value (right index stay same) else assing mid to divide the array into half
      #ifdef XDEBUG
            printf("mid: %d left: %d right: %d\n", mid[i], left[i], right[i]);
      #endif

      //int64_t flag = (data[mid[i]] > targets[i]);  //data[mid] >= target ? 1 : 0
      // left[i] = ((flag-1) & (mid[i]+1)) | ((flag^1)-1 & left[i]);
      // right[i] = ((flag-1) & right[i]) | ((flag^1)-1 & mid[i]);
      //break;
    }
    cnt+=1;
  }
}


/* The following union type is handy to output the contents of AVX512 data types */
union int8x4 {
  __m256i a;
  int64_t b[4];
};

void printavx(char* name, __m256i v) {
  union int8x4 n;

  n.a=v;
  printf("Value in %s is [%ld %ld %ld %ld ]\n",name,n.b[0],n.b[1],n.b[2],n.b[3]);
}

/*
 * Optinal for using AVX-512

  union int8x8 {
    __m512i a;
    int64_t b[8];
  };

  void printavx512(char* name, __m512i v) {
    union int8x4 n;

    n.a=v;
    printf("Value in %s is [%ld %ld %ld %ld %ld %ld %ld %ld ]\n",name,n.b[0],n.b[1],n.b[2],n.b[3]);
  }

 */

//#define SIMDDEBUG
inline void low_bin_nb_simd(int64_t* data, int64_t size, __m256i target, __m256i* result)
{
  /* this binary search variant
     (a) does no comparisons in the inner loop by using bit masking operations instead
     (b) doesn't require an exact match; instead it returns the index of the first key >= the search key.
         That's good in a DB context where we might be doing a range search, and using binary search to
         identify the first key in the range.
     (c) If the search key is bigger than all keys, it returns size.
     (d) does 4 searches at the same time using AVX2 intrinsics

     See https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-8/intrinsics-for-avx2.html
     for documentation of the AVX512 intrinsics

     Note that we're using the result array as the "right" elements in the search, and that searchkey is being passed
     as an __m256i value rather than via a pointer.
  */

 __m256i aleft = _mm256_set1_epi64x(0);
  __m256i aright = _mm256_set1_epi64x(size);
  __m256i amid;

  __m256i ones = _mm256_set1_epi64x(1);
  __m256i amask;
  __m256i datavec;

#ifdef SIMDDEBUG
  printf("low_bin_nb_simd\n");
#endif
  //AVX Intrinsic Guide: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
  __m256i cmp = _mm256_cmpgt_epi64(aright, aleft);
  //Check if all elements of vector is 0 using testz: https://stackoverflow.com/questions/32072169/could-i-compare-to-zero-register-in-avx-correctly + https://stackoverflow.com/a/32120039
  //Check all elements in cmp are 0 (if value of cmp is 0 it means left > right else -1)
  while(_mm256_testz_si256(cmp, cmp) == 0){
    amid = _mm256_srli_epi64(_mm256_add_epi64(aleft, aright), 1); //get middle
    __m256i amid_one = _mm256_add_epi64(amid, ones); //get middle+1 vector
    //Extract matching values of data using AVX index vector using i64gather: https://stackoverflow.com/questions/51128005/what-do-you-do-without-fast-gather-and-scatter-in-avx2-instructions
    datavec = _mm256_i64gather_epi64((long long*)data, amid, sizeof(int64_t));

    // inlining failed error for >= oeprtion
    //__mmask8 flag = _mm256_cmpge_epi64_mask(datavec, target);
    //Combine > operation and = operation to have >= operation: https://www.splashlearn.com/math-vocabulary/greater-than-or-equal-to
    __m256i cmpgt = _mm256_cmpgt_epi64(datavec, target);
    __m256i cmpeq = _mm256_cmpeq_epi64(datavec, target);
    __m256i flag = _mm256_or_si256(cmpgt, cmpeq);

    //if-else using bitwise source: https://stackoverflow.com/questions/3798601/conditional-using-bitwise-operator
    aleft = _mm256_or_si256(_mm256_and_si256(flag, aleft),_mm256_andnot_si256(flag, amid_one)); // if falg == -1 assign left value (left index stays same) else assing mid+1 to divide the array into half.
    aright = _mm256_or_si256(_mm256_and_si256(flag, amid),_mm256_andnot_si256(flag, aright)); // if flag == -1 assign mid value to divide array into half else assign previous right value (rihgt index stays same)

#ifdef SIMDDEBUG
    printavx("amid", amid);
    printavx("flag", flag);
    printavx("aright", aright);
    printavx("aleft", aleft);
    printf("\n\n");
#endif
    //break;
    cmp = _mm256_cmpgt_epi64(aright, aleft);
  }
  //aright = _mm256_sub_epi64(aright, ones);
  *result = aright;
}

void bulk_bin_search(int64_t* data, int64_t size, int64_t* searchkeys, int64_t numsearches, int64_t* results, int repeats)
{
  for(int j=0; j<repeats; j++) {
    /* Function to test a large number of binary searches

       we might need repeats>1 to make sure the events we're measuring are not dominated by various
       overheads, particularly for small values of size and/or numsearches

       we assume that we want exactly "size" searches, where "size" is the length if the searchkeys array
     */
    for(int64_t i=0;i<numsearches; i++) {
#ifdef DEBUG
```

```
      printf("Searching for %ld...\n",searchkeys[i]);
#endif

      // Uncomment one of the following to measure it
      //results[i] = low_bin_search(data,size,searchkeys[i]);
      //results[i] = low_bin_nb_arithmetic(data,size,searchkeys[i]);
      results[i] = low_bin_nb_mask(data,size,searchkeys[i]);

#ifdef DEBUG
      printf("Result is %ld\n",results[i]);
#endif
    }
  }
}

void bulk_bin_search_4x(int64_t* data, int64_t size, int64_t* searchkeys, int64_t numsearches, int64_t* results, int repeats)
{
  register __m256i searchkey_4x;

  for(int j=0; j<repeats; j++) {
    /* Function to test a large number of binary searches using one of the 8x routines

       we might need repeats>1 to make sure the events we're measuring are not dominated by various
       overheads, particularly for small values of size and/or numsearches

       we assume that we want exactly "size" searches, where "size" is the length if the searchkeys array
     */
    int64_t extras = numsearches % 4;
    for(int64_t i=0;i<numsearches-extras; i+=4) {
#ifdef DEBUG
      printf("Searching for %ld %ld %ld %ld  ...\n",
             searchkeys[i],searchkeys[i+1],searchkeys[i+2],searchkeys[i+3]);
#endif

      // Uncomment one of the following depending on which routine you want to profile

      // Algorithm A
      //low_bin_nb_4x(data,size,&searchkeys[i],&results[i]);

      // Algorithm B
      searchkey_4x = _mm256_loadu_si256((__m256i *)&searchkeys[i]);
      low_bin_nb_simd(data,size,searchkey_4x,(__m256i *)&results[i]);

#ifdef DEBUG
      printf("Result is %ld %ld %ld %ld  ...\n",
            results[i],results[i+1],results[i+2],results[i+3]);
#endif
    }
    /* a little bit more work if numsearches is not a multiple of 8 */
    for(int64_t i=numsearches-extras;i<numsearches; i++) {

      results[i] = low_bin_nb_mask(data,size,searchkeys[i]);

    }

  }
}


int64_t band_join(int64_t* inner, int64_t inner_size, int64_t* outer, int64_t outer_size, int64_t* inner_results, int64_t* outer_results, int64_t result_size, int64_t bound)
{
  /* In a band join we want matches within a range of values.  If p is the probe value from the outer table, then all
     reccords in the inner table with a key in the range [p-bound,p+bound] inclusive should be part of the result.

     Results are returned via two arrays. outer_results stores the index of the outer table row that matches, and
     inner_results stores the index of the inner table row that matches.  result_size tells you the size of the
     output array that has been allocated. You should make sure that you don't exceed this size.  If there are
     more results than can fit in the result arrays, then return early with just a prefix of the results in the result
     arrays. The return value of the function should be the number of output results.

  */
 //Declaring arrays to store left and right index of the range.
  int64_t *leftIndexArray = malloc(outer_size * sizeof(int64_t));
  int64_t *rightIndexArray = malloc(outer_size * sizeof(int64_t));
  if (!leftIndexArray || !rightIndexArray) {
    free(leftIndexArray);
    free(rightIndexArray);
    // Memory allocation failed, so we return -1
    return -1;
  }
  //Calcuating below values to find the multiplier and remainder.
  int multiplier = outer_size/4;
  int remainder = outer_size%4;

  int temp_1=0;

  while (temp_1<multiplier){
    // printf("Inside  muliplier with i->%d\n",temp_1);
    //using the below arrays to store the lower bound values temporarily after using low_bin_nb_mask function.
    int64_t lower_limit[4];
    int64_t upper_limit[4];

    for (int i =0;i<4;i++){
      lower_limit[i]=outer[4*temp_1+i]-bound;
      upper_limit[i]=outer[4*temp_1+i]+bound;
    }
    int64_t left_temp_Array[4];
    int64_t right_temp_Array[4];

    low_bin_nb_4x(inner,inner_size,lower_limit,left_temp_Array);
    low_bin_nb_4x(inner,inner_size,upper_limit,right_temp_Array);

    for (int i =0;i<4;i++){
      leftIndexArray[4*temp_1 + i]=left_temp_Array[i];
      rightIndexArray[4*temp_1 + i]=right_temp_Array[i];
    }
    // printf("Ending  muliplier with i->%d\n",temp_1);
    temp_1+=1;

  }

  for(int i = 0; i<remainder;i++){
    // printf("Entering remainder with i->%d\n",i);
    // printf("lower-> %d\n",outer[i]-bound);
    // printf("upper-> %d\n",outer[i]+bound);
    leftIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner, inner_size, outer[4*temp_1 + i]-bound);//array containing left index in the range.
    rightIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner,inner_size,outer[4*temp_1 + i]+bound);//array containing right index of the range.
    // printf("leftindex->%d \t\n rightindex->%d\t\n",leftIndexArray[i],rightIndexArray[i]);
    // printf("Exiting reminder loop with i->%d\n",i);
  }printf("\n");



/* ONLY WITH low_bin_nb_mask
  for(int i = 0; i<outer_size;i++){
    // printf("lower-> %d\n",outer[i]-bound);
    // printf("upper-> %d\n",outer[i]+bound);
    leftIndexArray[i]=low_bin_nb_mask(inner, inner_size, outer[i]-bound);//array containing left index in the range.
    rightIndexArray[i]=low_bin_nb_mask(inner,inner_size,outer[i]+bound);//array containing right index of the range.
    // printf("leftindex->%d \t\n rightindex->%d\t\n",leftIndexArray[i],rightIndexArray[i]);
  }printf("\n");
*/
// for(int i = 0; i<outer_size;i++){
// printf("leftIndexArray %d->%d and rightIndexArray %d -
>%d\n",i,leftIndexArray[i],i,rightIndexArray[i]);
// }

  int temp=0;

  for(int i = 0; i<outer_size;i++){
    // printf("Entering inside the merging part with i->%d\n",i);
    int left_index = leftIndexArray[i];
    int right_index = rightIndexArray[i];
    //if left index and right index are same then we skip the iteration because there are no elements.
    if (left_index==right_index){continue;}

    while ((temp<result_size) && (left_index<right_index)){
      // printf("Inside while for outer element index %d\n",i);
      outer_results[temp]=i;
      inner_results[temp]=left_index;
      // printf("adding values (%d,%d) in outer and inner result\n",outer_results[i],inner_results[i]);
      left_index+=1;
      temp+=1;

    }
//break if we reach the result size.
    if (temp==result_size){break;}
    // printf("Exiting merging part with i->%d\n",i);

  }

  // for (int i = 0; i<temp;i++){
  //   printf("(OuterResult,InnerResult)->(%ld,%ld)\n",outer_results[i],inner_results[i]);
  // }

  free(leftIndexArray);
  free(rightIndexArray);

  return temp;

}

int64_t band_join_simd(int64_t* inner, int64_t inner_size, int64_t* outer, int64_t outer_size, int64_t* inner_results, int64_t* outer_results, int64_t result_size, int64_t bound)
{
  /* In a band join we want matches within a range of values.  If p is the probe value from the outer table, then all
     reccords in the inner table with a key in the range [p-bound,p+bound] inclusive should be part of the result.

     Results are returned via two arrays. outer_results stores the index of the outer table row that matches, and
     inner_results stores the index of the inner table row that matches.  result_size tells you the size of
```

```
the
    output array that has been allocated. You should make sure that you don't exceed this size.  If there are
    more results than can fit in the result arrays, then return early with just a prefix of the results in the result
    arrays. The return value of the function should be the number of output results.

    To do the binary search, you could use the low_bin_nb_simd you just implemented to search for the lower bounds in parallel

    Once you've found the lower bounds, do the following for each of the 4 search keys in turn:
      scan along the sorted inner array, generating outputs for each match, and making sure not to exceed the output array bounds.

    This inner scanning code does not have to use SIMD.
  */



 //Using similiar approach from the above function.

  // Create and allocate space to store left and right result indexes
  int64_t *leftIndexArray = malloc(outer_size * sizeof(int64_t));
  int64_t *rightIndexArray = malloc(outer_size * sizeof(int64_t));
  if (!leftIndexArray || !rightIndexArray) {
    free(leftIndexArray);
    free(rightIndexArray);
    return -1;
  }

  // Calculate multiplier and remainder to find number of simd function instances and individual mask function call instances
  int multiplier = outer_size/4;
  int remainder = outer_size%4;

  int temp_1=0;
  while (temp_1<multiplier){
    // compute lower and upper bound vectors using AVX functions
    __m256i outer_vec = _mm256_loadu_si256((__m256i*)&outer[4*temp_1]);
    __m256i lower_bound_vec = _mm256_sub_epi64(outer_vec, _mm256_set1_epi64x(bound));
    __m256i upper_bound_vec = _mm256_add_epi64(outer_vec, _mm256_set1_epi64x(bound));

    // Create temporary indexes to store each iteration's results
    int64_t left_indexes[4];
    int64_t right_indexes[4];

    // Call binary search function using simd
    low_bin_nb_simd(inner,inner_size,lower_bound_vec,(__m256i*) left_indexes);
    low_bin_nb_simd(inner,inner_size,upper_bound_vec,(__m256i*) right_indexes);

    // Store results in temporary arrays
    for (int i =0;i<4;i++){
      leftIndexArray[4*temp_1 + i]=left_indexes[i];
      rightIndexArray[4*temp_1 + i]=right_indexes[i];
    }
    temp_1+=1;

  }

  // Store indexes of remaining cases into the result arrays
  for(int i = 0; i<remainder;i++){
    leftIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner, inner_size, outer[4*temp_1 + i]-bound);//array containing left index in the range.
    rightIndexArray[4*temp_1 + i]=low_bin_nb_mask(inner,inner_size,outer[4*temp_1 + i]+bound);//array containing right index of the range.
  }

  // temp will be the final result size
  int temp=0;

  for(int i = 0; i<outer_size;i++){
    int left_index = leftIndexArray[i];
    int right_index = rightIndexArray[i];

    if (left_index==right_index){continue;}

    // Store indexes into the result format
    while ((temp<result_size) && (left_index<right_index)){
      outer_results[temp]=i;
      inner_results[temp]=left_index;
      left_index+=1;
      temp+=1;

    }

    if (temp==result_size){break;}

  }

  // Clear index array space
  free(leftIndexArray);
  free(rightIndexArray);

  // Reuturn result size
  return temp;


 /*
```

```
  INITIAL CODE WHICH WAS GIVING INCONSISTENT RESULT. PLEASE SKIP TO END OF COMMENT FOR THE FINAL CODE.
 int size = 0;

  for(int i=0; (i<outer_size) && (size<result_size); i+=4){
    __m256i outer_vec = _mm256_loadu_si256((__m256i*)&outer[i]);
    __m256i lower_bound_vec = _mm256_sub_epi64(outer_vec, _mm256_set1_epi64x(bound));
    __m256i upper_bound_vec = _mm256_add_epi64(outer_vec, _mm256_set1_epi64x(bound));

    int64_t left_indexes[4];
    int64_t right_indexes[4];

    int remainder = outer_size - i;
    if (remainder >= 4) {

      __m256i left_result[1];
      __m256i right_result[1];

      low_bin_nb_simd(inner,inner_size,lower_bound_vec,left_result);
      low_bin_nb_simd(inner,inner_size,upper_bound_vec,right_result);

      _mm256_store_si256((__m256i*)left_indexes, left_result[0]);
      _mm256_store_si256((__m256i*)right_indexes, right_result[0]);

      for(int j = 0; j < 4; j++) {
        while((size < result_size) && (left_indexes[j] < right_indexes[j])) {
          outer_results[size] = i+j;
          inner_results[size] = left_indexes[j];
          left_indexes[j]++;
          size++;
        }
      }

    } else {

      if (remainder >= 1) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 0);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 0);

        left_indexes[0] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[0] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      if (remainder >= 2) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 1);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 1);

        left_indexes[1] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[1] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      if (remainder == 3) {
        int64_t lower_bound = _mm256_extract_epi64(lower_bound_vec, 2);
        int64_t upper_bound = _mm256_extract_epi64(upper_bound_vec, 2);

        left_indexes[2] = low_bin_nb_mask(inner,inner_size,lower_bound);
        right_indexes[2] = low_bin_nb_mask(inner,inner_size,upper_bound);
      }

      for(int j = 0; j < remainder; j++) {
        while((size < result_size) && (left_indexes[j] < right_indexes[j])) {
          outer_results[size] = i+j;
          inner_results[size] = left_indexes[j];
          left_indexes[j]++;
          size++;
        }
      }

    }

  }
  return size;
  */

}


```

▾ makefile Download

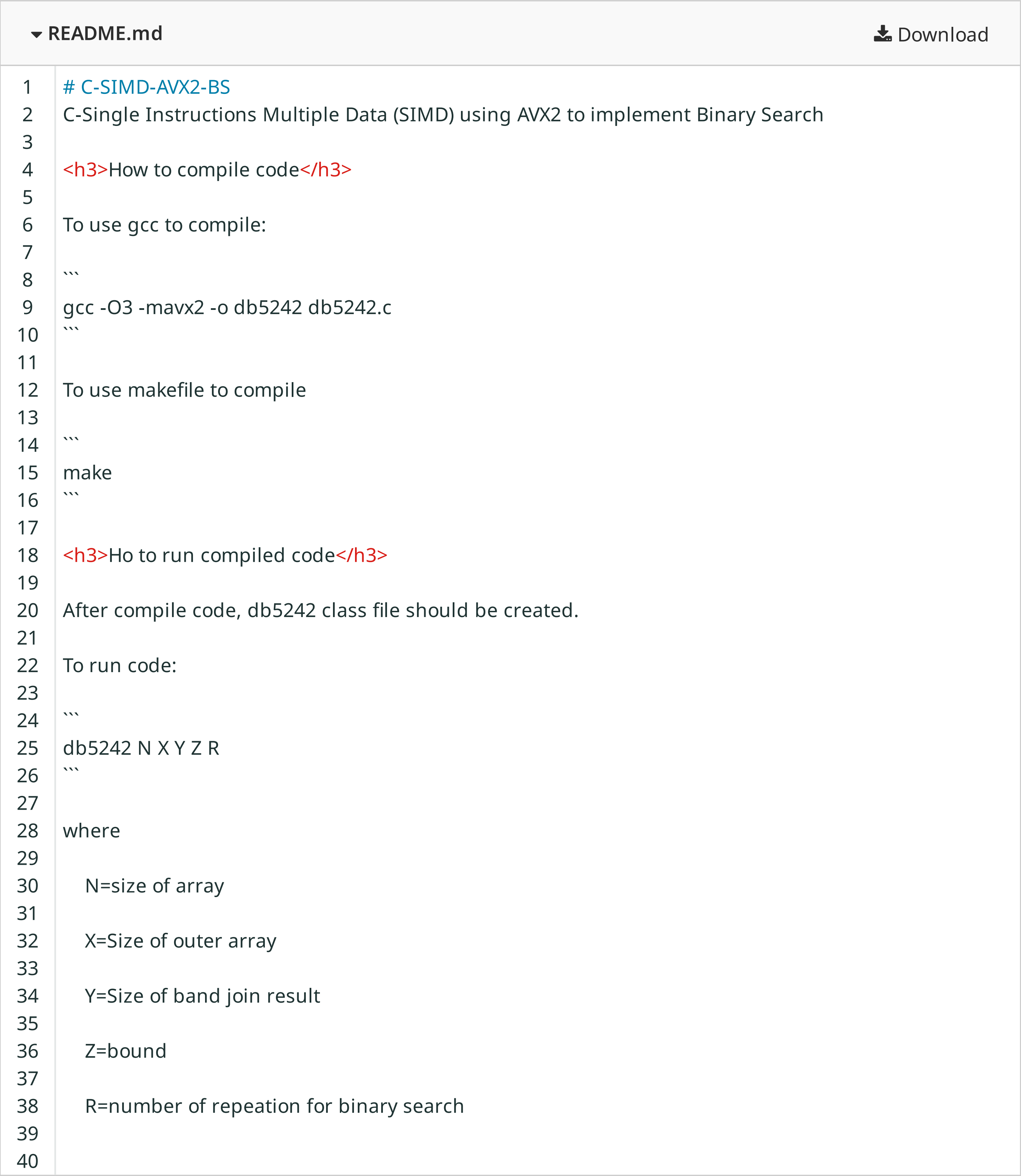
```
all: db5242

db5242: db5242.c
	gcc -O3 -mavx2 -o db5242 db5242.c

```

Project2 Document.pdf Download

Your browser does not support PDF previews. You can download the file instead.

Project2 wo grammarly (3).docx Download

Large file hidden. You can download it using the button above.



```
# C-SIMD-AVX2-BS
C-Single Instructions Multiple Data (SIMD) using AVX2 to implement Binary Search

<h3>How to compile code</h3>

To use gcc to compile:

```
gcc -O3 -mavx2 -o db5242 db5242.c
```

To use makefile to compile

```
make
```

<h3>Ho to run compiled code</h3>

After compile code, db5242 class file should be created.

To run code:

```
db5242 N X Y Z R
```

where

    N=size of array

    X=Size of outer array

    Y=Size of band join result

    Z=bound

    R=number of repeation for binary search


```