1: Pseudo Code for Algorithm

(a)

Algorithm 1 Build Suffix Array

```
1: function SA(s)                                   ▷ String s as input
2:     satups← [empty array]
3:     for i in |s| do
4:         Add s[i:|s|] to satups
5:     end for
6:     sort the satups based on lexicographically order
7:     SA[sort index] gets original suffix index
8:     return SA
9: end function
```

(b)

Algorithm 2 Convert Suffix Array To BWT

```
1: function BWT(sa, text)                   ▷ Suffix array sa and the text as input
2:     bwt← [empty array]
3:     for i in |sa| do
4:         Add text[sa[i] - 1] to bwt
5:     end for
6:     return bwt
7: end function
```

(c)

Algorithm 3 Construct FM index's OCC

```
1: function RANK(bwt)                               ▷ BWT bwt as input
2:     tots ← new dict
3:     ranks ← new array
4:     for c in bwt do
5:         if c not in tots then
6:             let tots[c]←0
7:         end if
8:         Add tots[c] to ranks
9:         tots[c] ←tots[c]+1
10:    end for
11:    return ranks, tots
12: end function
```

Algorithm 4 Construct FM index's C

```
1: function COUNT(tots)                                   ▷ tots as input
2:     totc ← 0
3:     first ← new array
4:     sort tots with its key
5:     for c, count in tots do
6:         first[c]← totc
7:         totc ← totc + count
8:     end for
9:     return first
10: end function
```

(d)

Algorithm 5 Update search range

```
1: function UPDATE(pi, start, end)  ▷ Character pi, row index start, and row index end as input
2:     start← ranks(pi, start - 1) + first(pi)
3:     end← ranks(pi, end) + first(pi) - 1
4:     return start, end
5: end function
```

Algorithm 6 Check if reach the end

```
1: function CHECK(start, end)                  ▷ ow index start, and row index end as input
2:     range← suffixarray[start: end+1]
3:     for index in range do
4:         if text[index - 1] = $ then
5:             return index
6:         end if
7:     end for
8: end function
```

Algorithm 7 Find overlap between pattern's prefix and index' suffix

```
1: function UPDATE(p, fmindex, threshold)        ▷ Pattern p, FM index fmindex, and overlap threshold as input
2:     reverse input pattern p
3:     start← 0
4:     end← length of indexed text -1
5:     overlapsize← 0
6:     for c in reversed p do
7:         start, end ← Update(c, start, end)
8:         overlapsize← overlapsize++
9:         if overlapsize ¿ threshold then
10:            Check(start, end)
11:            if Check return value then stop and return the overlap size
12:            end if
13:        end if
14:    end for
15: end function
```

2: Running Time

On HPCC, it took about 40 seconds using 1 thread to finish the all against all compare on the given HIV reads dataset

3: Memory Usage

For the all against all compare on the given HIV reads dataset, my program use 131 MB memory