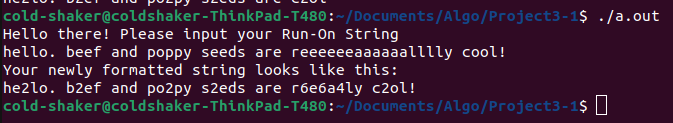
3

**Algorithm 1: String Run Encoding**

*Source code for this project can be found in the accompanying project file stringrun.cpp.*

| **Pseudocode:** | **Complexity Analysis** |
| --- | --- |
| def **a** = 0 //anchor  def **p** = 1 //parser  def **m** = 1 // numbers of matching chars  def string unencoded\_string  def string encoded\_string  ask user for input  insert user input –>> unencoded\_string  while a < length of unencoded\_string  if unencoded\_string[p] == unencoded\_string[a]  while unencoded\_string[p] == unencoded\_string[i]  increment m  increment p  append **m** to encoded\_string  append **a** to encoded\_string  def **a** = **p**  def **m** = 1  else  append **a** to encoded\_string  increment **a**  increment **p**  output encoded\_string | O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(n)  O(1)  O(n)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1) |
| Total efficiency: 3n^2 + 10n + 9 | = O(n^2) efficiency class |

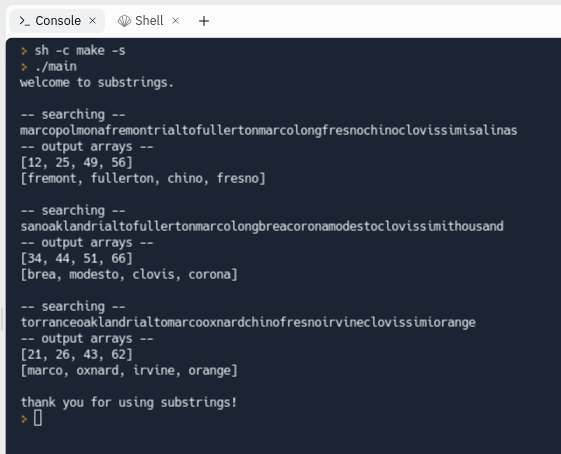
****

**Algorithm 2: Finding Target Substrings**

*Source code for this project can be found in the accompanying project file alg2.cpp.*

| **Pseudocode:** | **Complexity Analysis** |
| --- | --- |
| def find\_substrings(str1, str2[]):  for each **i** in str2[]  def **found** false  def counter **k**  for each **j** elements in str2[**i**]  while traversing str1 && substring !**found**  if str2[**i**][**j**] element == str1 element **k**  def f**index[i]** -> **k**  def counter **c** =1  def counter **h** = 0  def counter **l == k + 1**  while traversing str2[**i**]  if str2[**i**][**j+1**] element == str1 element **l**  **j++**  **l++**  **c++**  if **c** == str2[**i**].size()  **found** true  **h++**  **k++** | O(1)  O(n)  O(1)  O(1)  O(n) // nested -> n^2  O(n) // nested -> n^3  O(1)  O(1)  O(1)  O(1)  O(1)  O(n) // nested ->n^4  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1) |
| Total efficiency: 8n^4 + 7n^3 + n^2 + 3n + 1 | = O(n^4) efficiency class |

**Algorithm 2: Finding Target Substrings**

**Output:**

**Algorithm 3: Merging Techniques**

*Source code for this project can be found in the accompanying project file alg3.cpp.*

| **Pseudocode:** | **Complexity Analysis** |
| --- | --- |
| def merge\_sorted lists(lists[]):  def vector sort\_all[]  insert list[**0**] -->> sort\_all[]  for each **i** in lists[]  for each **j** elements in lists[**i**]  def **k** element in sort\_all  def **placed** false    while traversing sort\_all **k** && element **j** !**placed**  if lists[**i**][**j**] element < sort\_all element **k**  insert lists[**i**][**j**] -->> front of sort\_all  **placed** = true    if list[**i**][**j**] > all elements of sort\_all  insert lists[**i**][**j**] -->> end of sort\_all  **placed** = true  **k**++ | O(1)  O(1)  O(1)  O(n)  O(n) // nested -> n^2  O(1)  O(1)  O(n) // nested ->n^3  O(1)  O(1)  O(1)  O(1)  O(1)  O(1)  O(1) |
| Total efficiency: 8n^3 + 3n^2 + n + 3 | = O(n^3) efficiency class |

**Algorithm 3: Merging Techniques**

**Output:**

