# Assignment 1

## Task 1 (a) – Rough Algorithm



## Task 1 (a) – Final Algorithm

**Algorithm** isOrderedShuffle(a, b, c)

**If** (c = a+b)

**Return** true

**Endif**

**If** (c.length != a.length +b.length)

**Return** false

**Endif**

**If** (c.length = 0)

**Return** true

**Endif**

**If**(a[0]=b[0])

**If**(a.length **or** b.length = 1)

**If** (a[0]=c[0])

**Return** isOrderedShuffle(a[1 to end], b, c[1 to end])

**Endif**

**If** (b[0]=c[0])

**Return** isOrderedShuffle(a, b[1 to end], c[1 to end])

**Endif**

**Endif**

**If** (a!=b)

i := 0

**While** a[0 to i] = a[0 to i]

i++

**EndWhile**

**If** (a[0 to i]=c[0 to i])

**Return** isOrderedShuffle(a[i to end], b, c[i to end])

**Endif**

**If** (b[0]=c[0])

**Return** isOrderedShuffle(a, b[i to end], c[i to end])

**Endif**

**Endif**

**Endif**

**If** (a[0]=c[0])

**Return** isOrderedShuffle(a[1 to end], b, c[1 to end])

**Endif**

**If** (b[0]=c[0])

**Return** isOrderedShuffle(a, b[1 to end], c[1 to end])

**Endif**

**Return** false

## Task 1 (b)

**Best Case:**

Best case is *c length != a length + b length*

O(1)

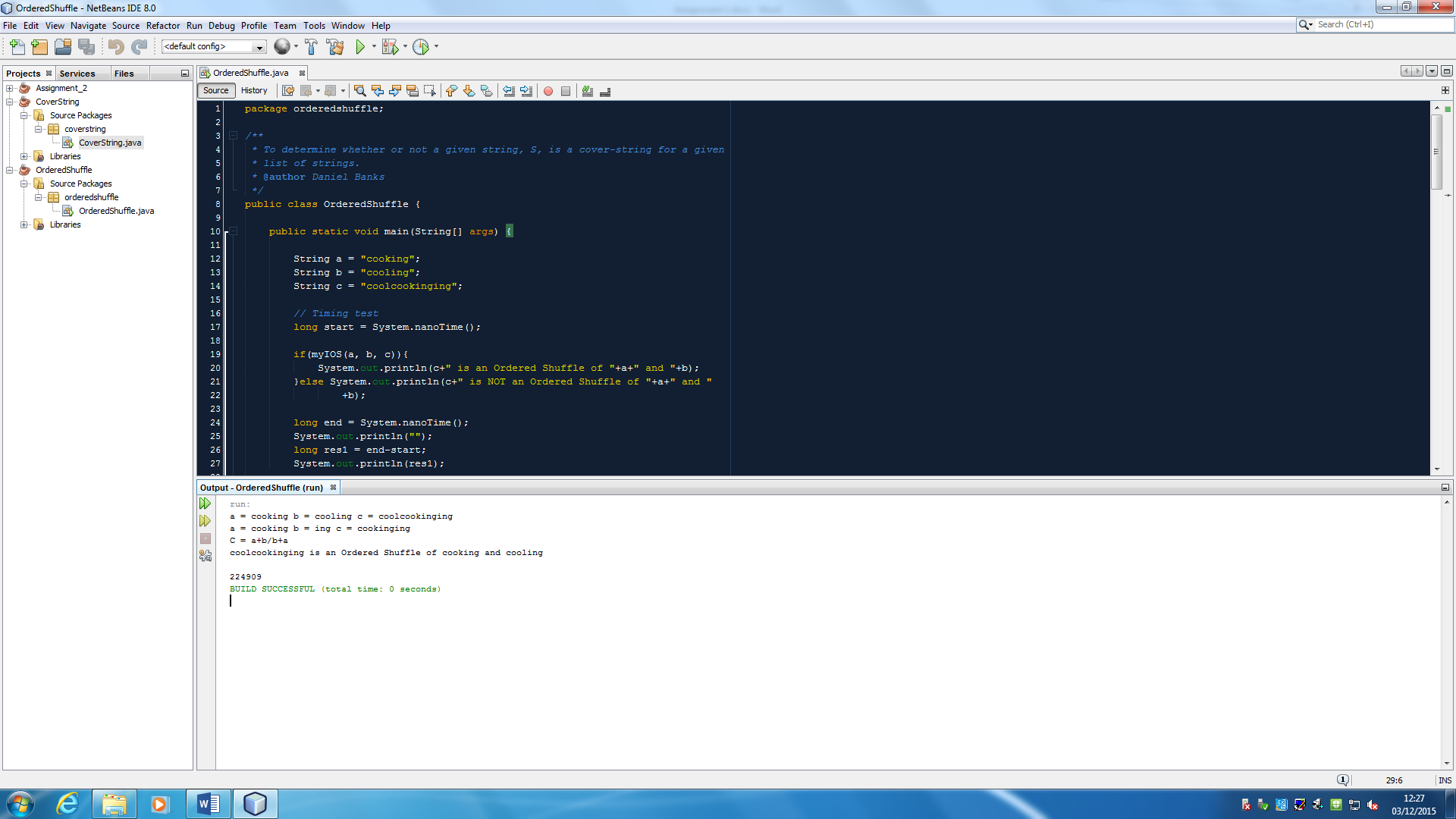
**Worst Case:**

Worst case is having to search entire c

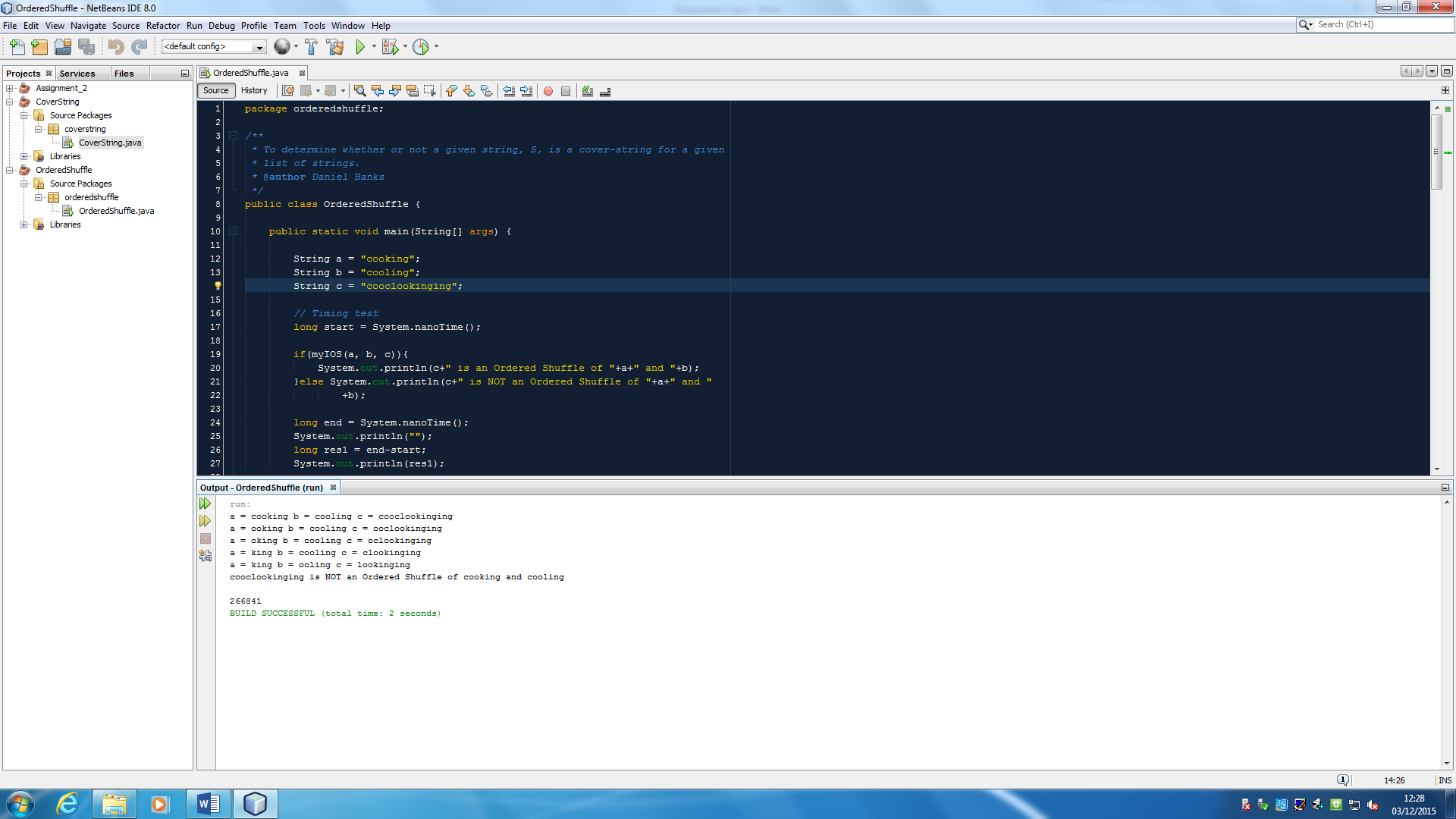
O(n)

## Task 1 – Testing

**Pass case**



**Fail case**



## Task 2 (a) – Rough Algorithm



## Task 2 (a) – Final Algorithm

**Algorithm** isCoverString(StringList, CoverString)

**For Each** String in StringList

Int a = 0

Boolean cover = false

Int i = 0

**For** each letter in CoverString

**If** String[a to a+1] := CoverString[I to i+1]

a++

**if** a := String.Length

cover = true

**Return** cover

## Task 2 (b)

Worst case would be having to search the entire cover string for each string in the list.

n = cover string

k = number of strings in list

O(nk)

## Task 2 (c)

The best way to generate a worst case instance of my algorithm would be to remove the break statement in the main isCoverString method, and then create a method to generate a cover string that is always correct. This way the algorithm will read through the entire cover string for every string in the list. And because the cover sting is correct, it will have to check every string.

Another way to generate a pretty bad, but not worst, case for my algorithm would be to generate the cover string by concatenating all of the strings in the list, excluding the last letters, then put all of the last letters at the end of the cover string. This way would not require any code being commented out like the one above, but it will not create an absolute worst case.

## Task 2 (d)

Ran for 2^1 up to 2^20, and m = 4, 8 & 16.

As the timing results below show, when the size of k increases the time it takes the algorithm to check the cover string increases by around x3, up to 2^12, then it increases to around x4 the time to complete. Then as the size m (string length) increases, the time roughly doubles from the test before.

String Length = 4

|  |  |  |  |
| --- | --- | --- | --- |
| **length** | **Repetitions** | **elapsedTime** | **timePerSearch** |
| 2 | 1099962 | 1000000604 | 909 |
| 4 | 441502 | 1000000302 | 2265 |
| 8 | 139241 | 1000005735 | 7182 |
| 16 | 44974 | 1000000302 | 22235 |
| 32 | 16215 | 1000009962 | 61672 |
| 64 | 6336 | 1000006339 | 157829 |
| 128 | 2654 | 1000223990 | 376874 |
| 256 | 1175 | 1000521939 | 851508 |
| 512 | 516 | 1001168552 | 1940249 |
| 1024 | 203 | 1000921620 | 4930648 |
| 2048 | 73 | 1006765587 | 13791309 |
| 4096 | 25 | 1019456646 | 40778266 |
| 8192 | 8 | 1105751684 | 138218961 |
| 16384 | 2 | 1015742999 | 507871500 |
| 32768 | 1 | 2097879430 | 2097879430 |
| 65536 | 1 | 8102252670 | 8102252670 |
| 131072 | 1 | 32410626306 | 32410626306 |
| 262144 | 1 | 126852422248 | 126852422248 |
| 524288 | 1 | 579922608416 | 579922608416 |
| 1048576 | 1 | 2430650048738 | 2430650048738 |

String Length = 8

|  |  |  |  |
| --- | --- | --- | --- |
| **length** | **Repetitions** | **elapsedTime** | **timePerSearch** |
| 2 | 709820 | 1000001208 | 1409 |
| 4 | 249385 | 1000000604 | 4010 |
| 8 | 81558 | 1000010264 | 12261 |
| 16 | 24278 | 1000030791 | 41191 |
| 32 | 8659 | 1000032904 | 115491 |
| 64 | 3253 | 1000140371 | 307452 |
| 128 | 1405 | 1000506544 | 712104 |
| 256 | 612 | 1000785173 | 1635270 |
| 512 | 256 | 1000398171 | 3907805 |
| 1024 | 109 | 1006059505 | 9229904 |
| 2048 | 40 | 1012995347 | 25324884 |
| 4096 | 12 | 1032359620 | 86029968 |
| 8192 | 4 | 1063951276 | 265987819 |
| 16384 | 1 | 1007187606 | 1007187606 |
| 32768 | 1 | 3918763152 | 3918763152 |
| 65536 | 1 | 15858012641 | 15858012641 |
| 131072 | 1 | 77417417854 | 77417417854 |
| 262144 | 1 | 389883138167 | 389883138167 |
| 524288 | 1 | 1702205969728 | 1702205969728 |
| 1048576 | 1 | 7182368540627 | 7182368540627 |

String Length = 16

|  |  |  |  |
| --- | --- | --- | --- |
| **length** | **Repetitions** | **elapsedTime** | **timePerSearch** |
| 2 | 379734 | 1000000000 | 2633 |
| 4 | 136874 | 1000001510 | 7306 |
| 8 | 44586 | 1000015697 | 22429 |
| 16 | 13834 | 1000013886 | 72287 |
| 32 | 3841 | 1000088449 | 260372 |
| 64 | 1596 | 1000639368 | 626967 |
| 128 | 686 | 1000600124 | 1458601 |
| 256 | 304 | 1001721584 | 3295137 |
| 512 | 113 | 1007973080 | 8920116 |
| 1024 | 51 | 1017457339 | 19950144 |
| 2048 | 19 | 1019896476 | 53678762 |
| 4096 | 6 | 1036255599 | 172709267 |
| 8192 | 2 | 1212436459 | 606218230 |
| 16384 | 1 | 2237705678 | 2237705678 |
| 32768 | 1 | 9288240793 | 9288240793 |
| 65536 | 1 | 39915561181 | 39915561181 |
| 131072 | 1 | 186159119882 | 186159119882 |
| 262144 | 1 | 840883702670 | 840883702670 |
| 524288 | 1 | 3646558470311 | 3646558470311 |
| 1048576 | 1 | 12018154494578 | 12018154494578 |

BUILD SUCCESSFUL (total time: 489 minutes 1 second)