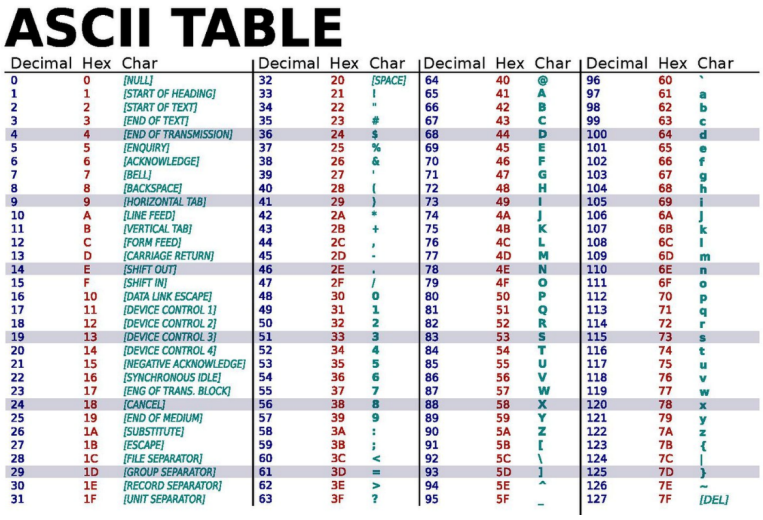
Assignment 3

1. Quicksort

Implement the Quicksort algorithm to sort an array of lowercase characters. The selection of the pivot element is based on the following process:

1. Calculate the ASCII value for each character in the array.
2. Compute the sum of these ASCII values.
3. Divide the sum by the total number of elements in the array. If the result is a decimal, take the floor value (round down to the nearest integer).
4. Perform a modulo operation using the result from Step 3 with the total number of elements in the array.
5. The remainder obtained from this operation determines the index of the pivot element.

Example:
Input array: {f, m, n, t, r, k, q, w}
Array size: 8
ASCII values: 102 + 109 + 110 + 116 + 114 + 107 + 113 + 119 = 890
Average ASCII: 890 / 8 = 111.25
Floor value: floor(111.25) = 111
Pivot index: 111 % 8 = 7
Hence, the character at index 7 (w) is selected as the pivot.
Use a separate array of characters for testing your implementation.



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

2. Recursive Algorithm

Write a recursive program to partition a string s into all possible substrings such that every substring in a partition is a palindrome. The string s consists of lowercase English letters.

The output should be a list of lists, where each inner list represents a valid partition of the string into palindromic substrings. Ensure that all possible valid partitions are generated.

Examples:
Input: "aab"
Output: [["a", "a", "b"], ["aa", "b"]]

Input: "racecar"
Output:
```
[
  ["r", "a", "c", "e", "c", "a", "r"],
  ["r", "aceca", "r"],
  ["racecar"]
]
```

Requirements:

You are asked to test the code on the following cases:

1. Any string s that will list a palindrome partitions
2. You will need to find all palindromic partitions, but each palindrome in the partition must be at least 3 characters long. (ex. on abacdc - [["aba", "cdc"]])
3. You will need to find all palindromic partitions, but the number of partitions in each result should not exceed 2. (ex. on aabb - [["aabb"], ["aa", "bb"]])