# Florida Atlantic University (FAU)

## *Programming Assignment 1*


**Project objective**: Use Quicksort algorithm to sort sequences based off pseudocode, analyze computational running time for used partition methods and include code/report.

**Written By**: Kevin Tudor

# Table of Contents

# Introduction (aims/motivation)

This assignment is to learn how to code and analyze the computational running time of the quicksort algorithm with my chosen partition method. Upon understanding how the algorithm works, proper use in the future should occur.

## Assignment details:

Use **Quicksort algorithm** to sort the following sequences:

a. [10, 80, 3, 19, 14, 7, 5, 12]

b. Choose your sequence with 100 different (random) integer numbers.

c. Choose your sequence with 1000 different (random) integer numbers.

**Submission**: A single PDF with your code (Python programming language), results and analysis.

1. Please include your pseudocode, input sequence and output in the report.

2. Also, you need to analyze the computational running time for your partition methods (e.g., best/worst/average cases) theoretically. You ***may*** use the sequences b and c as the example to illustrate.

3. Report template: Introduction (aims/motivation), Background (review/research), Project Specification (program analysis, solution design), Implementation (evaluation), Summary (conclusions), Reference.

*Note: No information on correct title for single pdf submission.*

## Background (review/research)

The Quicksort algorithm is a **divide and conquer** algorithm (like Merge sort) that is utilized to efficiently sort elements in a particular order (other popular algorithms include Heapsort, and Bubble sort).

The Quicksort algorithm was first contributed to by Tony Hoare. Hoare required an algorithm that could assist in a project he was assigned to around 1960 and thus, the Quicksort algorithm was granted its initial publicized use/contribution.

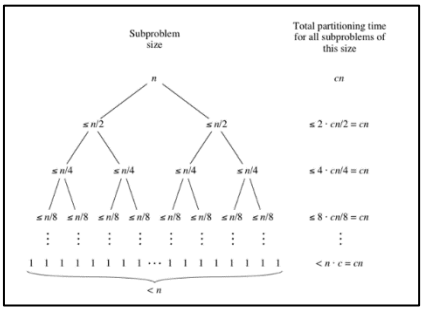
The Quicksort algorithm will first define "pivot" points at which it will split the list in smaller sub-arrays depending on whether they are larger or smaller than the "pivot point" (**Begin Divide**). It then continues to *recursively* split each subsection of the original list (**Divide**) until it reaches a "portion" that is *efficient* to sort (one or less elements - **Begin Conquer**). Upon reaching that point, the recursion will break, and the elements will be "swapped" and pushed back into ordered place (**Conquer**). When all recursions are broken and sorted, the algorithm can then return a sorted list.

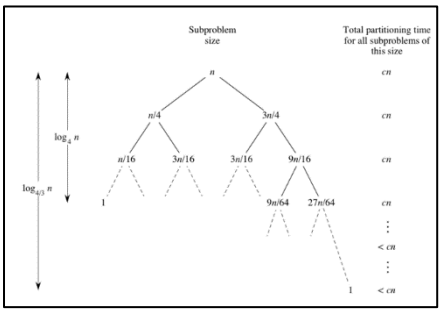
# Project Specification (program analysis, solution design)

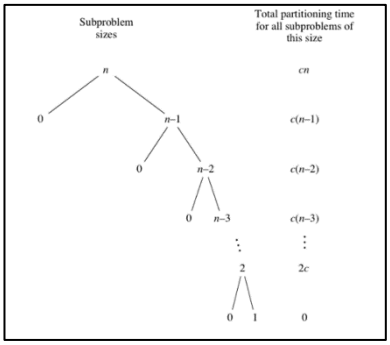
## Pseudocode:

```
QUICKSORT(A,p,r)
If p < r
        q = PARTITION(A,p,r)
        QUICKSORT(A,p,q-1)
        QUICKSORT(A,q+1,r)

PARTITION(A,p,r)
x = A[r]
i  = p-1
For j = p to r-1
        If A[j] <= x
                i = i+1
                exchange A[i] with A[j]
exchange A[i+1] with A[r]
return i + 1

RANDOMSEQUENCE(NUM)
Return list of “num” different random number/s ranging from 0 to num
```

## Computational Running Time:



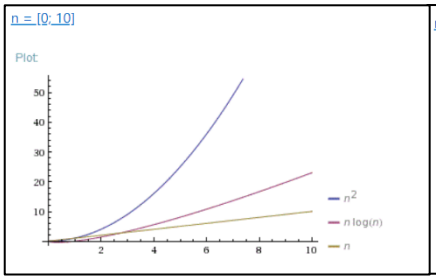



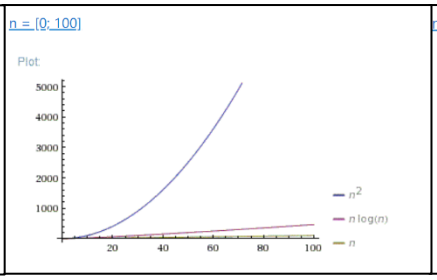



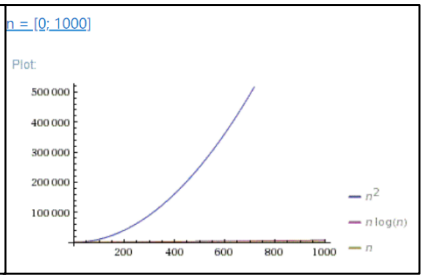

Best: $\Theta(n \lg_2 n)$ Average: $\Theta(n \lg_2 n)$ Worst: $\Theta(n^2)$

The theoretical computational running time is based on the chosen method of choosing "pivot points", the number of times needed to partition, size of the dataset, and if the dataset is oven/odd number of elements. For example, when the partition is recursively directly in the middle of an even list each recursive partition will ultimately compute an even fraction of the entire list. When the data is not split down the middle, the time can vary as the algorithm may recursively create more child nodes thus increasing the recursive complexity and computational time. The method used in the code below will not be the best case as the algorithm will pivot as last element.

Worst case will be $T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$ which summed up will evaluate $\Theta(n^2)$. Best case will be when data is split most equally and $T(n) = 2T(n/2) + \Theta(n)$ which summed up will equal $\Theta(n \lg_2 n)$. Average case (9 to 1 split) example would be $T(n) = T(9n/10) + T(n/10) + cn$ which summed up equals $\Theta(n \lg_2 n)$.




When the dataset is very small, the "worst" time may be close to the "best" time but as the amount of element to be sorted increases, the "best" time is far more optimal. Thus, with larger data sets, it is important to choose the optimal partition methods (middle pivot) when using quicksort.

# Implementation (evaluation)

## Code:

```
# -*- coding: utf-8 -*-
"""
IDE: Spyder (Python 3.8)
@author: Kevin Tudor
Course: COT 4400: Design and Analysis of Algorithms
Programming 1: Quicksort Algorithm
Due: March 13, 2023 (Monday), 11:59PM
"""
# include libraries
import random

def quicksort(a, p, r):
    if p < r:
        q = partition(a, p, r)
        quicksort(a, p, q - 1)
        quicksort(a, q + 1, r)

def partition(a, p, r):
    x = a[r]
    i = p - 1

    for j in range(p, r):  # for j = p to r - 1:
        if a[j] <= x:
            i = i + 1
            # swap
            temp = a[i]
            a[i] = a[j]
            a[j] = temp
    # swap
    temp2 = a[i + 1]
    a[i + 1] = a[r]
    a[r] = temp2
    return i + 1

def rseq(num):
    return random.sample(range(0, num), num)
```

```
def main():
    # a. 10, 80, 3, 19, 14, 7, 5, 12
    a = [10, 80, 3, 19, 14, 7, 5, 12]
    print("Unsorted - GIVEN sequence:\n", a)
    quicksort(a, 0, len(a) - 1)
    print("\nSorted - GIVEN sequence: \n", a)

    # b. Choose your sequence with 100 different (random) integer numbers
    b = rseq(100)
    print(
        "\nUnsorted - sequence with",
        len(b),
        "different (random) integer numbers: \n",
        b,
    )

    quicksort(b, 0, len(b) - 1)
    print(
        "\nSorted - sequence with", len(b), "different (random) integer numbers: \n", b
    )

    # c. Choose your sequence with 1000 different (random) integer numbers
    c = rseq(1000)
    print(
        "\nUnsorted - sequence with",
        len(c),
        "different (random) integer numbers: \n",
        c,
    )

    quicksort(c, 0, len(c) - 1)
    print(
        "\nSorted - sequence with", len(c), "different (random) integer numbers: \n", c
    )

if __name__ == "__main__":
    main()
```

## Most Recent Input/ Output:

**Unsorted - GIVEN sequence:**
[10, 80, 3, 19, 14, 7, 5, 12]

**Sorted - GIVEN sequence:**
[3, 5, 7, 10, 12, 14, 19, 80]

**Unsorted - sequence with 100 different (random) integer numbers:**
[57, 6, 27, 32, 56, 86, 41, 80, 51, 8, 88, 74, 90, 44, 13, 92, 5, 24, 75, 11, 79, 84, 50, 72, 33, 28, 87, 69, 9, 7, 48, 85, 77, 4, 43, 91, 76, 10, 0, 16, 81, 64, 67, 52, 46, 93, 99, 1, 2, 61, 82, 39, 96, 29, 71, 58, 14, 19, 94, 65, 40, 15, 38, 97, 25, 18, 66, 26, 49, 55, 95, 42, 36, 12, 60, 68, 62, 59, 34, 70, 20, 23, 22, 89, 45, 53, 83, 31, 30, 21, 37, 73, 3, 98, 78, 54, 35, 63, 17, 47]
**Sorted - sequence with 100 different (random) integer numbers:**
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]

**Unsorted - sequence with 1000 different (random) integer numbers:**
[205, 136, 618, 810, 371, 382, 121, 305, 282, 576, 521, 315, 607, 591, 161, 199, 327, 988, 312, 256, 538, 453, 150, 152, 644, 920, 710, 676, 661, 985, 474, 70, 903, 570, 21, 600, 892, 45, 167, 928, 527, 580, 619, 914, 244, 6, 18, 557, 426, 246, 39, 615, 32, 195, 102, 975, 729, 842, 28, 288, 731, 915, 608, 252, 587, 54, 812, 274, 348, 442, 652, 758, 177, 630, 989, 4, 929, 549, 806, 507, 621, 254, 697, 287, 412, 930, 128, 363, 20, 64, 217, 247, 492, 219, 101, 35, 300, 8, 837, 532, 317, 994, 134, 19, 153, 341, 144, 764, 673, 781, 36, 402, 216, 754, 306, 966, 483, 182, 949, 180, 784, 851, 935, 704, 229, 757, 386, 248, 943, 470, 320, 83, 40, 945, 429, 537, 181, 431, 922, 511, 592, 714, 374, 55, 555, 835, 990, 824, 964, 370, 692, 793, 263, 816, 340, 417, 542, 112, 586, 275, 653, 109, 642, 506, 82, 803, 440, 381, 577, 27, 7, 239, 539, 530, 88, 490, 245, 894, 209, 207, 461, 34, 924, 795, 359, 783, 512, 843, 57, 719, 196, 830, 620, 560, 685, 322, 861, 958, 769, 280, 675, 627, 680, 395, 170, 154, 625, 441, 59, 595, 712, 849, 701, 838, 628, 97, 773, 261, 204, 186, 749, 137, 967, 953, 110, 438, 61, 93, 684, 476, 99, 983, 380, 269, 565, 865, 89, 850, 696, 63, 400, 50, 333, 132, 105, 188, 926, 434, 736, 566, 748, 873, 299, 917, 659, 569, 904, 801, 409, 194, 49, 358, 665, 148, 790, 973, 666, 654, 752, 831, 883, 360, 563, 26, 593, 233, 885, 65, 811, 242, 550, 293, 69, 200, 925, 637, 215, 42, 415, 759, 477, 707, 571, 647, 292, 986, 852, 187, 439, 159, 100, 214, 390, 725, 411, 346, 328, 906, 671, 735, 761, 574, 120, 876, 262, 554, 51, 221, 786, 176, 385, 94, 882, 695, 389, 91, 747, 422, 107, 663, 794, 92, 398, 962, 471, 367, 979, 948, 494, 828, 259, 770, 854, 533, 785, 191, 516, 342, 936, 869, 179, 562, 60, 115, 118, 746, 283, 635, 178, 941, 479, 497, 750, 640, 887, 343, 447, 999, 523, 880, 817, 603, 683, 788, 720, 157, 942, 366, 797, 255, 354, 755, 868, 578, 825, 392, 473, 753, 30, 491, 413, 787, 460, 502, 871, 111, 961, 330, 291, 531, 266, 912, 728, 295, 250, 845, 226, 10, 658, 891, 391, 564, 171, 237, 174, 125, 501, 495, 238, 698, 349, 760, 270, 745, 679, 104, 820, 331, 24, 844, 634, 944, 145, 289, 866, 674, 613, 528, 138, 3, 303, 192, 48, 957, 80, 827, 839, 208, 267, 995, 706, 970, 103, 584, 631, 5, 230, 468, 778, 939, 717, 833, 993, 633, 551, 846, 590, 241, 260, 808, 558, 169, 443,

383, 896, 656, 489, 636, 689, 552, 599, 959, 895, 974, 332, 485, 427, 505, 352, 419, 691, 743, 131, 940, 963, 774, 938, 826, 933, 780, 117, 927, 863, 172, 905, 639, 212, 253, 316, 321, 308, 301, 597, 629, 548, 664, 932, 662, 160, 403, 575, 968, 41, 609, 475, 908, 235, 855, 201, 937, 143, 323, 857, 632, 127, 155, 782, 907, 776, 573, 339, 375, 954, 878, 1, 325, 910, 46, 510, 123, 841, 297, 950, 992, 298, 319, 206, 307, 437, 732, 243, 450, 139, 796, 462, 156, 918, 222, 190, 672, 66, 56, 822, 37, 996, 984, 889, 733, 734, 163, 751, 730, 766, 336, 198, 913, 124, 146, 302, 612, 326, 384, 556, 596, 711, 185, 872, 543, 202, 805, 832, 601, 220, 660, 286, 309, 583, 353, 670, 972, 976, 324, 338, 79, 362, 594, 231, 455, 582, 779, 345, 414, 678, 708, 271, 223, 424, 493, 898, 81, 829, 12, 463, 0, 53, 998, 798, 807, 646, 165, 126, 257, 767, 874, 113, 740, 408, 357, 604, 875, 705, 459, 762, 690, 416, 184, 723, 681, 702, 486, 623, 166, 519, 520, 481, 31, 738, 189, 982, 373, 406, 741, 572, 335, 234, 433, 264, 151, 687, 513, 568, 641, 273, 792, 142, 211, 227, 848, 763, 87, 579, 888, 955, 467, 946, 62, 657, 545, 454, 598, 540, 478, 716, 724, 290, 909, 279, 258, 162, 488, 853, 364, 534, 638, 47, 203, 432, 430, 645, 368, 164, 9, 667, 278, 881, 651, 25, 388, 183, 16, 529, 669, 668, 509, 141, 484, 197, 147, 775, 296, 13, 344, 240, 76, 524, 655, 643, 418, 856, 436, 836, 313, 772, 585, 119, 514, 446, 379, 173, 739, 624, 804, 224, 448, 840, 158, 393, 567, 991, 890, 546, 969, 95, 456, 605, 68, 700, 458, 369, 814, 879, 175, 72, 236, 272, 472, 713, 304, 899, 265, 737, 559, 777, 616, 43, 135, 466, 78, 85, 727, 372, 544, 951, 611, 499, 971, 693, 819, 451, 960, 648, 277, 351, 965, 535, 726, 547, 541, 420, 149, 361, 602, 469, 686, 404, 71, 22, 73, 445, 122, 285, 210, 553, 897, 517, 232, 765, 96, 870, 449, 350, 744, 823, 457, 480, 756, 622, 815, 649, 337, 487, 791, 58, 329, 376, 500, 923, 859, 977, 688, 709, 423, 17, 394, 428, 218, 318, 425, 821, 378, 956, 397, 84, 522, 518, 310, 650, 677, 38, 114, 33, 813, 931, 606, 98, 90, 981, 334, 452, 435, 116, 515, 52, 864, 800, 503, 919, 193, 14, 268, 902, 228, 356, 399, 213, 934, 465, 771, 916, 699, 294, 980, 742, 525, 130, 900, 347, 718, 407, 721, 581, 715, 901, 311, 129, 314, 11, 893, 847, 249, 276, 464, 799, 29, 884, 355, 626, 133, 77, 74, 405, 498, 377, 997, 284, 444, 140, 561, 921, 2, 23, 818, 789, 834, 867, 610, 365, 858, 589, 978, 860, 496, 703, 387, 768, 952, 108, 588, 86, 225, 396, 802, 44, 614, 67, 168, 947, 862, 526, 722, 877, 617, 694, 251, 536, 886, 504, 421, 911, 508, 987, 410, 682, 281, 401, 482, 15, 75, 809, 106]

**Sorted - sequence with 1000 different (random) integer numbers:**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370,

371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999]

## Summary

By successfully implementing and analyzing the Quicksort algorithm, a deeper and more meaningful understanding of the algorithm should provide insight on future use. Quicksort is not the only algorithm that can used to sort elements in a particular order (other popular algorithms include Merge sort, Heapsort, and Bubble sort). Thus, depending on the size of the dataset and the complexity of the "sort" it is important to understand the chosen algorithm and its running time (efficiency) then it is valid to use the correct algorithm.

In reflection, the pseudocode had initial issues when transcribing in python. Some ranges needed to be corrected and the code would only work properly for a list of 8 elements or less (lists of length > 8 would only sort the first half partition and this error may have been due to a mistake in transcribing the second recursive call of QUICKSORT in the QUICKSORT function).

*Note: python can create a random sequence with random.sample and I used it to create a random list of unique numbers from 0 to the given number then used quicksort to sort it in ascending order. Thus sorted arrays are 0 – 999 max.*

# References

**Book:**

Cormen, T. H., & Leiserson, C. E. (2009). CH 7 - Quicksort. In *Introduction to algorithms, 3rd Edition* (pp. 150–174).

**Basic Info on Quicksort:**

GeeksforGeeks. (2023, March 7). *Quicksort*. GeeksforGeeks. Retrieved March 13, 2023, from https://www.geeksforgeeks.org/quick-sort/

**Theoretical Quicksort Tree diagrams:**

Khan Academy. (n.d.). *Analysis of Quicksort (article) | quick sort*. Khan Academy. Retrieved March 13, 2023, from https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

**Dataset size vs Time diagrams:**

*Which is better: O(n log N) or o(n^2)*. Stack Overflow. (1961, March 1). Retrieved March 13, 2023, from https://stackoverflow.com/questions/23329234/which-is-better-on-log-n-or-on2

**Initial public Contribution:**

*Quicksort*. Tony Hoare >> Contributions >> Quicksort. (n.d.). Retrieved March 13, 2023, from https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/tony-hoare/quicksort.html