Assignment #2

Ali Akbari
30010402

Notes:
Words like list and array are used interchangeably.
Pictures of recursion trees/charts are self-made.
Sources are given for material used.

Identities used:

- If $f(n) = \mathcal{O}\left(n^{\log_b(a-c)}\right)$ for some $c \in \mathbb{R}_{>0}$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
- If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = n^{\log_b a} \log n$.
- If $f(n) = \Omega\left(n^{\log_b(a-c)}\right)$ for some $c \in \mathbb{R}_{>0}$, and $af(\frac{n}{b}) \leq kf(n)$ for some $k < 1$ and n is sufficiently large, then $T(n) = \Theta\left(f(n)\right)$.

Quick Sort

1) **Description:**
   A divide and conquer algorithm which takes a problem, and simplifies it to a set of smaller problems. A sorting algorithm that sorts by choosing a pivot point. Partition the collection around the pivot, elements smaller than the pivot are before the pivot, and the element larger are after the pivot. Through recursion break the lists into the single-element list before combing them back together for a sorted list. Given a variable-sized array consisting of numbers, run the following recursive algorithm:
   1 - Select a “splitting value” index known as the pivot from the array. For this algorithm select the element at the end element of the array as the pivot.
   2 - Select a Low variable to point to the lowest index.
   3- Select the High variable as the last index excluding the pivot.
   4- Move the Low variable to the right/ increment left by 1 while the value of the array at Low is less than the pivot.
   5- Move the High variable to the left/ decrement left by 1 while the value of the array at High is greater than the pivot.
   6- If conditions for steps 4 & 5 do not meet, swap Low element and High element.
   7- If the Low variable index is greater or equal to the High variable index, the point where they met is the new pivot position.
   8- Return the left sub-array without the pivot and quicksort the subarray(Recursive step, calls the quick sort algorithm as many times as the subarray needs).
   9- Return the right sub-array without the pivot and quicksort the subarray(Recursive step, calls the quick sort algorithm as many times as the subarray needs).
   10- Merge the array back together and return the sorted Array.

The pseudocode referenced from GeeksforGeeks:

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
   if (low < high)
   {
      /* pi is partitioning index, arr[pi] is now
         at right place */
      pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }
}
```

2) **Average Case**

T(n) = aT(n/b) + f(n)

$a \geq 1,\ b > 1\ and\ a,\ b \in R$

Base Case:

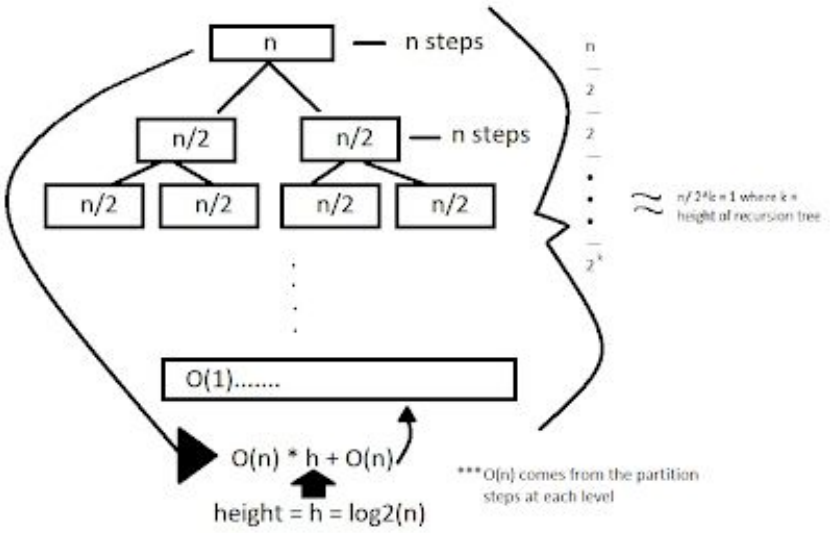
When the number of elements of the array is less than 2, the element does not need to be sorted thus, order of big-O of constant.

n∘ < 2

T(0), T(1) = O(1)

Assuming the pivot is chosen at mid and the partition is done at the middle of the array/list,

Let n > n∘ then:



From the recursion tree, we get the height from $n/2^k = 1$ since we are dividing the elements by 2, till it reaches 1 element we can simplify and get: $n = 2^k \rightarrow k = log_2(n)$. This means that there are log n levels.
At each level, we get that the partition part of the algorithm which is comparing and replacing each element as such it goes through the entire array which gives a time complexity of $O(n)$.

By using our identities, it falls into the second condition
a = 2 b = 2 $f(n) = \mathrm{O}(n^{log_b(a)}) \rightarrow f(n) = \mathrm{O}(n^{log_2(2)}) \rightarrow f(n) = \mathrm{O}(n)$
$T(n) = n^{log_b{}^a} log(n) \rightarrow T(n) = n^{log_2{}^2} log(n) \rightarrow T(n) = n^1 log(n)$
Therefore $T(n) = \mathrm{O}(n) * \mathrm{O}(log_2(n)) + \mathrm{O}(n)$
So time complexity of the average case of quicksort is:
$T(n) = O(log_2(n))$

3) **Run Times**
    a) n = 10
       = 0.0 seconds
    b) n = 100
       = 0.0029964447021484375 seconds
    c) n = 1000
       = 0.007984638214111328 seconds

Merge Sort

1) **Description**

A divide and conquer algorithm which takes a problem, and simplifies it to a set of smaller problems. It divides input array into two halves, repeatedly calls itself for the two halves until singleton lists remain. Then merge the singleton lists in ascending order. Given a variable-sized array consisting of numbers, run the following recursive algorithm:

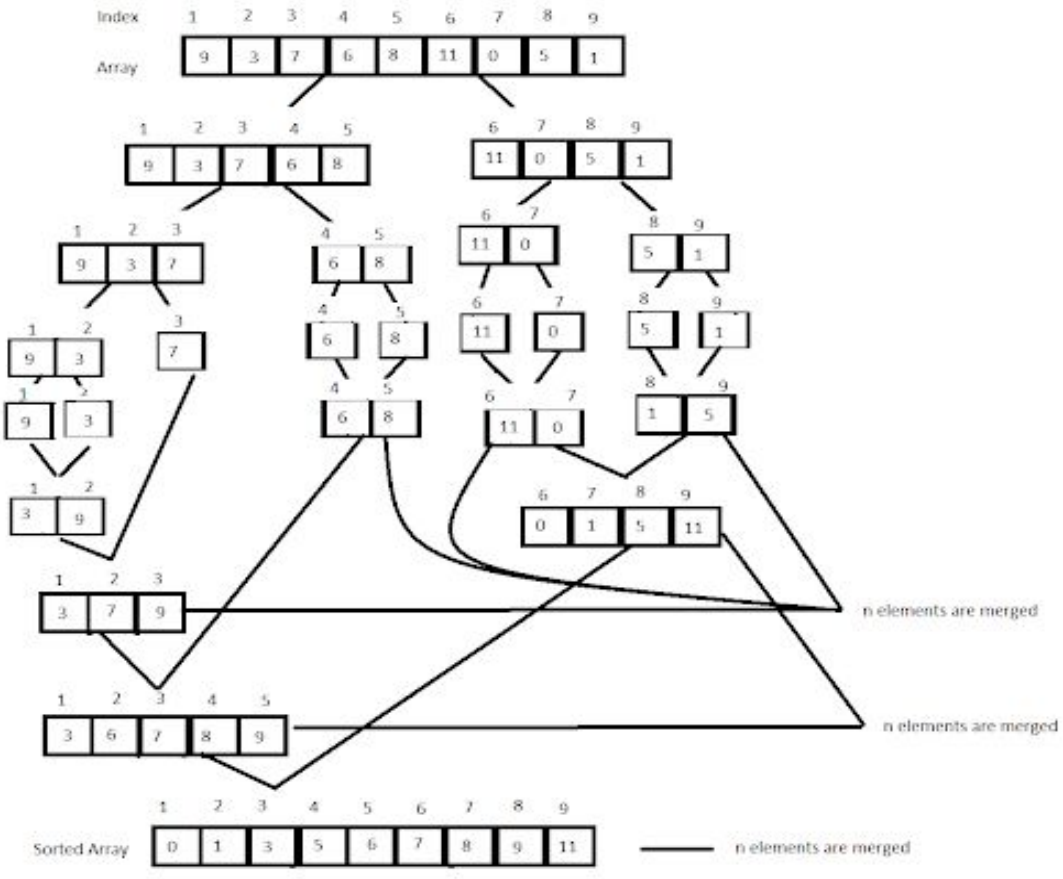
1) Find the middle of the array. Divide the array in the middle.
2) Call algorithm for the left half. Call algorithm for the right half.
3) Call recursively for both steps 1 & 2, until arrays of size 1 remain.
4) Take adjacent pairs of two singleton lists and merge (compare and sort) them to form a list of the 2 elements.
5) Take adjacent pairs of two doubleton lists and merge (compare and sort) them to form a list of the 4 elements.
6) Repeat the process until a single sorted list is obtained.

Pseudocode:

```
Algorithm mergeSort(low, high):          #T(n)
        middle = (low + high)// 2        # cost of 1
        low = array[0 to mid]            #
        high = array[mid to length of array]
        If low < high:
        mergeSort(low, middle)           #T(n/2)
        mergeSort(middle +1, high)       #T(n/2)
        merge(low, middle, high)         #n ---a function that sorts and merges
                                         #the two arrays
        return array
```

2) Average Case

T(n) = aT(n/b) + f(n) $\quad a \geq 1,\ b > 1\ and\ a,\ b \in R$



Base Case:
When the number of elements of the array is less than 2, the element does not need to be sorted thus, order of big-O of constant.
n∘ < 2
T(0), T(1) = O(1)

Let n > n∘ then:
From the recursion tree, we get that there are $log_2(n)$ levels. We get the height from $n/2^k = 1$ since we are dividing the elements by 2, till it reaches 1 element we can simplify and get: $n = 2^k \rightarrow k = log_2(n)$. This means that there are $log_2(n)$ levels. We see that there are 3 levels of merging from 9 element array, and at each level we have n elements merging. The merge part of the algorithm which is comparing and replacing each element as such it goes through the entire array which gives a time complexity of $O(n)$.
From the pseudocode we get:
$T(n) = 2T(n/2)$ + n

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

Our variables are defined as:
a = 2
b = 2
f(n) =n

To see which case it falls into:
$f(n) = \mathrm{O}(n^{log_b(a)}) \rightarrow f(n) = \mathrm{O}(n^{log_2(2)}) \rightarrow f(n) = \mathrm{O}(n)$
$log_b{}^a = log_2{}^2 = 1$
$n^k = n^1$
$log_2{}^2 = k$
So by comparing these two we get that this falls in case two from our identities which follows with:
$T(n) = n^{log_b{}^a} log(n) \rightarrow T(n) = n^{log_2{}^2} log(n) \rightarrow T(n) = n^1 log(n)$
$\mathrm{O}(n * log_2(n)$
So time complexity of the average case of mergesort is:
$T(n) = \mathrm{O}(n * log_2(n))$

3) **Run Times**
    a) n = 10
       = 0.0 seconds
    b) n = 100
       = 0.003998756408691406 seconds
    c) n = 1000
       = 0.0160064697265625 seconds

Binary Search

1) **Description:**

A recursive divide and conquer algorithm that takes a problem, and simplifies so that it is faster to compute a search. Finds the midpoint of an array A, compares it to a key v (the value we are looking for). If the key is equal to midpoint return the midpoint index. Else if the key is less than midpoint, repeat steps for the left half of the array. Else look for the right half of the array. Once a single element is left on the half that meets the condition (left half = key < midpoint / right half = key > midpoint) return the element index if the key is the same as the element. Given a variable-sized array consisting of numbers, run the following recursive algorithm:

1) For an element v to be found in a sorted array compute the middle of the array.
2) Compare v to the middle. If middle matched the element returns the middle index.
3) Else if less than the middle element look to the left subarray. Else if greater than middle element, look to the right subarray.
4) Repeat steps 2 & 3, return the index if a match is found.

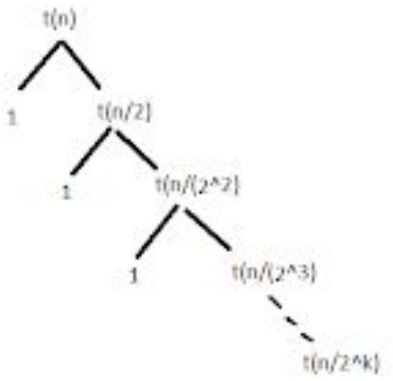
Recursive Pseudocode From Rosetta Code:

```
// initially called with low = 0, high = N-1
BinarySearch(A[0..N-1], value, low, high) {
    // invariants: value > A[i] for all i < low
                   value < A[i] for all i > high
    if (high < low)
        return not_found // value would be inserted at index "low"
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid
}
```

2) **Average Case**

Pseudocode

```
Algorithm binSearch (low, high, key) ---------- T(n)
        if low == high:                 ---------- 1  ##
                if key == A[low]:       ---------- 1  ## Base Case
                        Return low      ---------- 1  ##
                else:                                 ##
                        Return -1       ---------- 1  ##
        else:
                middle = (low + high) //2 ---------- 1
                if mid == A[middle]:    ---------- 1
                        return middle   ---------- 1
                if mid < A[middle]:     ---------- 1
                        return binSearch(low, middle - 1, key)        --- T(n/2)
                if mid > A[middle]:
                        return binSearch(middle + 1, high, key)       --- T(n/2)
```



Base Case:
From our Pseudocode we get that when the number of elements of the array is less than or equal to 1, the element either is return or -1 is returned, order of big-O of constant.
n∘ < 1
T(0), T(1) = O(1)
c = constant

$T(n) = \{ 1 \quad n = 1$

$T(n/2) + c \quad n > 1$ --Either calls left hand side or right hand side

From this we get that:

a = 1

b = 2

f(n) =n

And from the second image of the recursive tree we get that the algorithm will process through the array of $t(n) = n/2^k$ and from the Pseudocode we can see that this will stop once we have 1 element.

So $n/2^k = 1$

The time complexity depends on the levels of the recursive tree as there is only a constant amount of work/steps done at each level, so the levels are determined by the equation above. K = levels which also the time complexity of binary search.

To find k:

$n/2^k = 1$

$2^k = n$

$k = log_2 n$

To see which case it falls into:

$f(n) = \mathrm{O}(n^{log_b(a)}) \rightarrow f(n) = \mathrm{O}(n^{log_2(1)}) \rightarrow f(n) = \mathrm{O}(n)$

$log_b{}^a = log_2{}^1 = 0$

So by comparing these two we get that this falls in case two from our identities which follows with:

$T(n) = n^{log_b{}^a} log(n) \rightarrow T(n) = n^{log_2{}^1} log(n) \rightarrow T(n) = n^0 log(n)$

$\mathrm{O}(1 * log_2(n)$

So time complexity of the average case of binary search is:

$T(n) = \mathrm{O}(log_2(n))$

3) Run Times

a) n = 10
   First = 0.0 seconds
   n/2 = 0.004000186920166016 seconds
   n = 0.003999233245849609 seconds

b) n = 100
   First = 0.0 seconds
   n/2 = 0.0 seconds
   n = 0.003999233245849609

c) n = 1000
First = 0.0 seconds
n/2 = 0.004006385803222656 seconds
n = 0.003999471664428711

Inputs for my runtime:

n = 10

2451.08 9788.07 -4598.73 3875.8 6923.91 -1770.92 6790.17 -1662.43 3264.61 -3805.91

n = 100

1674.99 -3551.49 -6116.03 1015.97 2914.14 6355.63 -6819.48 -7715.91 -9464.69 -7884.66 -658.43 -4473.95 4687.82 -3604.95 -7329.57 -3723.69 -1059.57 -659.05 3934.32 -846.17 -1242.82 3373.65 -3233.21 6902.3 127.63 -3659.13 4053.88 3386.79 -7367.81 2087.53 -6021.32 -5043.01 1372.12 -7172.52 -3329.4 4417.61 7272.09 -6793.15 2912.98 1056.87 2300.61 807.1 -508.82 -2540.97 733.17 4756.03 -5712.35 665.94 2996.78 7780.16 -739.93 5178.37 -1295.37 -5267.75 -4056.72 -8693.7 -2577.11 8470.62 -5537.67 -194.65 3962.37 -7936.96 -7009.32 -1419.37 -2837.55 858.18 1103.77 2036.58 -1876.01 3905.9 4726.33 -3794.73 9401.24 4996.29 745.64 3099.64 -812.27 -1514.15 -1074.71 8782.93 -2162.88 -7145.21 -9926 1416.94 -4891.28 3926.23 2074.94 4701.17 -4170.13 -6339.3 -535.85 -3934.27 1797.01 -1032.45 -1823.26 3070.55 -3843.35 -2139.79 3604.83 -1768.97

n = 1000

-8458.67 -1502.35 -1470.49 -4376.78 -997.06 703.03 -3614.94 -1177.33 2151.14 5091.67 -582.14 5775.33 164.15 -1697.6 4338.73 8173.9 -1520.54 6331.35 2974.22 7109.41 1559.91 4225.88 5218.04 -1183.67 -2787.08 113.7 -1752.55 -4608.47 5532.61 2912.52 1323.83 375.52 665.97 -9168.69 -1319.07 2340.89 -9430.41 2891.98 1974.66 -8427.42 -9800.4 -6459.3 3764.72 4377.68 -5222.09 9827.79 8087.33 -4842.79 -2370.72 -2020.13 -2699.3 -3788.79 3429.3 3377.26 3231.26 145.73 -8819.01 1673.25 -643.55 -5723.46 633.25 -1568.33 -1379.21 3757.74 -2993.29 6055.16 -6423.3 608.15 254.46 -4423.14 6442.3 1686.67 4141.85 -2994.26 -1477.08 90.41 4265.38 7937.08 -8044.23 -5516.78 8765.25 2542.17 -7946.59 4175.78 7406.64 8609.42 4378.71 1416.43 1319.91 2548.21 6692.71 -6560 -7416.84 -3952.36 -3864.12 2665.38 -3844.73 7509.64 -4009.36 -7145.8 -759.69 599.45 -771.2 -7487.74 3267.83 -5158.43 1919.99 -3000.05 -52.35 5661.19 4025.16 -1977.27 -596.34 -3061.14 5277.27 7540.99 2223.66 -9860.51 -7714.26 -1164.37 2823.28 -1371.94 2346.05 -4580.59 8862.67 2759.44 -4361.35 -6174.52 5015.66 857.1 8848.53 -4071.1 -2162.22 -773.2 2461.43 -2581 -672.93 -2242.23 2350.68 3613.09 5470.95 -2673.02 8940.96 6032.75 -5494.03 -3269.19 3753.62 -1835.87 2507.87 -2926.93 315.2 -4370.12 4188.33 -1233.99 -1606.88 2721.14 7829.79 -2883.98 -6434.16 -4988.41 2514.26 -1986.08 4721.21 3779.42 7767.27 523.74 6317.84 -5017.55 -2317.9 -5115.9 7508.92 6306.42 -9112.52 -7852.7 4645.04 7121.6 -4873.14 -3802.3 489.02 7736.45 -3522.23 3987.19 -356.75 4257.7 -2571.76 -7234.83 2084.49 -479.4 9025.01 -2932.24 6437.33 -2433.74 3282.1 -2435.96 123.58 -6463.88 -4145.12 3480.01 -7430.05 3656.18 -1010.94 -3030.71 -3636.92 -2278.14 1195.59 538.81 9375.91 2445.27 1154.78 -2609.83 5318.32 6115.27 -2437.28 -1804.19 -3752.9 392.05 -3313.45 -3830.28 5932.13 -301.49 1323.78 2449.79 937.3 -688.28 7754.38 -2730.28 -1704.98 -5588.09 6822.46 10.51 -6360.74 -4472.88 -40.55 -5310.59 -7134.81 4305.46 -1494.62 -1499.52 1606.59 678.5 1299.6 -974.03 -5675.82 -6049.07 5471.4 8228.94 8199.86 6811.36 820.06 -1299.61 -1469.11 -2155.75 -1245.02 4017.28 -3179.05 -1554.1 6633.41 -1349.42 7954.69 -8211.58 -6553.04 -6288.02 2223.27 -546.06 2415.62 -1499.4 4091.13 814.68 -1129.42 809.87 4232.31 -592.45 7374.44 -2167.33 -4972 1477.83 -3455.96 3232.8 3902.22 -3321.39 -7144.66 808.36 -5732.15 -3390.23 -1981.25 6262.59 3458.96 -6247.45 -4147.17 5428.68 -5731.55 -2333.69 4731.96 797.68 -3434.63 -3.73 8768.52 6791.08 -63.08 -8171.78 6497.1 2921.32 3767.86 -1772.04 2452.73 7753.76 148.76 2997.16 3081.11 -7469.42 -5079.25 2487.78 1610.33 3238.85 5207.04 6190.41 -291.67 6208.83 8181.74 2597.53 8921.76 -5694.17 -7641.16 5508.28 1811.55 1995.82 -3952.52 -7943.97 -9278.11 1160.78 1569.19 2094.49 2784.78 673.44 -1943.94 -664.24 553.63 -7267.59 7277.73 -527.58 -7004.89 -2891.45 -3870.72 1405.48 3624.23 -3549.43 3990.27 -5376.2 -1409.09 475.12 -8853.87 -4372.51 -2010.22 2039.74 7876.39 -9354.88 450.9 -4537.4 -2487.39 2857.71 7718.58 1015.24 -7339.54 7671.63 4019.56 -744.55 -4694.34 -8081.61 -1658.04 4258.59 -8745.49 3908.78 -55.21 -4090.61 -2261.16 8303.18 -7263.07 -4340 268.25 4082.6 1984.72 2619.15 3614.29 4975.93 5986.8 -9188.03 335.98 -2657.6 -3089.19 -4239.89 -455.04 -1692.78 1740.89 -7096.78 3737.65 -444.06 -5912.75 5498.46 5681.77 2016.64 5398.48 -1731.98 7530.4 -2680.98 -673.32 2303.52 -3223.14 -3452.14 236.83 -170.01 -1566.55 1813.11 -4150.88 2119.7 -2588.12 1827.16 -2126.59 3695.95 -56.09 -1663.44 4353.87 892.91 1347.29 -1173.51 -9428.86 -4928.1 -8746.73 246.62 -20 1466 -6496.33 -6938.53 1663.02 170.01 -2800.79 6131.17 -4273.69 497.19 3759.77 -3234.72 -57.95 687.02 6275.38 -787.92 -6612.05 -2066.07 -2092.55 9007.8 2087.25 4389.85 -7202.31 -2278.59 5450.74 -3244.78 4211.14 -8793.79 -7259.11 -1549.78 1434.82 -95.1 -5768.51 -1115.57 916.12 78.99 9201.17 3254.4 -7957.72 -4967.24 424.07 -7059.14 -5115.29 5684.49 -8328.01 -2324.25 3741.88 5558.75 -8719.27 -382.56 -5591.54 -4757.88 1559.66 2697.92 -9538.04 8348.09 7402.85 6866.15 7560.79 5747.37 7429.77 6149.85 7063.14 -3656.67 -8969.31 -832.84 6098.8 2466.68 1731.54 -6476.2 9371.52 1030.1 1038.06 660.57 -7533.68 3299 2565.7 1313.89 4213.44 -54.18 -2015.83 9382.56 7772.18 -4503.84 -1649.79 4545.87 -3627.07 1664.27 1076.92 9434.24 -1483.37 -3614.62 -2660.33 -2536.51 1545.19 7602.68 -5248.42 -3152.73 -8144.42 4585.25 2486.96 3669.28 -1190.18 6240.31 5441.44 192.03 -524.6 -1072.52 983.29 6877 2629.77 3179.9 -3036.63 -3855.5 -8065.93 -1414.56 4385.02 -326.02 -5334.28 647.3 -4096.24 6631.12 -1454.52 5037.92 2243.87 -1542.06 4757.59 -6264.97 3735.72 -173 -3371.28 6569.62 652.69 -3556.06 -2463.19 1834.38 230.15 -7899.25 9602.67 -2927.53 -4251.11 875.37 -1623.05 -8213.48 -404 9006.48 2215.21 5480.37 -1242.22 -430.37 741.49 80.05 -2791.04 -8386.99 5638.72 -3947.25 -3824.58 8363.1 -489.82 -796.25 -8610.19 -1655.58 4814.84 -2371.28 6152.29 9596.14 1267.89 4149.71 2315.06 2000.04 884.12 3440.93 -1136.51 3486.3 4692.38 900.38 -896.08 1820.67 -8319.96 9268.12 -6533.29 2805.54 -3045.51 -2040.17 -5631.03 711.97 -7498.94 6040.41 -701.94 -547.51 -467.04 -2911.43 7966.01 -7059.09 -4514.81 -359.92 -6692.59 -5025.06 9810.03 -7146.35 -1219.09 4066.04 -4866.8 -1726.29 9755.13 2259.5 -9104.42 -2275.83 6587.5 -3961.14 -5124.77 -6144.16 -3070.86 1213.21 2159.56 -6526.38 2382.59 1577.81 7299.94 -5590.78 -3945.17 -1928.51 5195.93 7453.47 4698.18 670.15 -4012.09 -5493.56 -7457.7 -265.73 -2904.97 -4690.44 -293.16 -2698.52 -1914.96 -5155.24 1355.1 5794.45 4623.59 5334.41 -2498.81 14.62 9982.35 -1918.63 6158.91 8435.01 -3343.19

-3481.06 1332.17 806.61 6253.25 91.77 -4262.44 -3945.18 2025.47 8250.88 4951.12 1918.6 3429.41 6609.73 -8223.2 155.18 -5516.52
-5001.58 -2188.7 3338.96 9333.37 2944.02 1563.17 9907.45 -7205.65 -2055.14 -479.41 -160.94 -2093.23 -4355.3 843.69 8857.47
-4945.22 -8412.29 -940.02 -9709.29 176.15 3671.95 1110.94 -5029.13 -1172.32 -803.48 9122.37 4650.07 -4098.9 358.25 6095.19
3233.83 -7019.31 -2907.91 -8976.3 -7185.32 -4874.13 -453.04 -1294.19 1121.37 -2082.37 3835.73 3423.09 -5427.7 -7791.37 3610.93
2590.44 4798.47 -7215.55 3726.29 -8779.34 8464.57 -7983.94 -1624.92 1141.43 -1046.27 -2041.08 4951.04 2464.29 -2007.63 -3181.77
-9445.97 -7272.65 1271.46 -3615.67 -8150.87 4160.26 9067.53 -9717.62 -4330.83 7619.18 -21.96 1221 -2815.61 -3338.73 -4802.44
-890.27 -6779.04 1075.61 957.98 -3312.47 -3824.44 2424.25 8350.3 8003.74 3330.24 -528.56 1733.79 -9929.36 -1313.38 5567.22
1013.78 5088.97 -742.04 207.6 -6801.32 8958.36 -2080.05 332.82 3153.04 3924.47 6308.8 -7551.97 -2805.35 6512.11 -62.36 1803.67
646.72 -612.83 -7542.5 1754.62 2556.91 6454.79 8036.56 1918.1 -5838.1 -6992.61 4488.89 -5278.15 259.12 -8588.14 -439.81 3434.99
-2839.46 226.23 -3012.48 -71.65 183.47 3906.66 -280.08 -1226.48 -7599.54 1882.67 2097.86 -4196.73 -1240.02 6780.97 -2481.46
6738.33 -229.66 873.67 -7413.17 -1154.65 -2736.94 -5328.23 -146.5 4386.68 2080.79 6441.31 -1025.77 -3400.6 -1551.11 -8511.81
-100.22 8695.74 2859.69 -1988.57 2094.85 -2789.61 -3649.58 -6257.89 -5017.07 9078.98 978.12 -9325.03 -2764 4290.86 2629.12
5959.02 -670.9 -139.16 2010.03 -1951.02 -7449.58 -6894.82 3383.03 3014.9 -924.98 7451.46 4800.07 2333.1 1652.66 -4337.83 -670.3
841.78 598.97 1139.01 9058.18 -2280.46 -4390.82 -1872.2 -1283.86 4044.66 -1019.37 5036.18 438.64 6109.97 -3370.79 9605.51
-7022.26 -3522.39 1296.64 -784.79 1291.8 -8614.6 5222.98 -6546.17 6062.45 2948.76 -6173.31 -167.18 -4161.7 -6399.59 -250.58
-7261.25 3741.82 2646.65 7533.27 -5918.72 5401.08 1172.55 4221.03 4148.37 -6089.37 -2671.02 2683.65 -7836.83 -1252.88 5968.43
-2611.07 -2225.57 6924.28 -843.69 2567.36 -7883.77 781.69 387.64 -6861.74 4549.83 4908.02 4123.72 -733.46 -1441.82 2543.76
-7779.35 -6065.84 -3682.29 -714 1810.06 7439.98 -4981.59 -479.91 -1831.02 -2680.63 6150.72 7024.23 -1846.75 347.65 7809.84
4885.35 -7937.25 1574.95 905.13 5834.15 1515.63 -9271.19 -4713.82 -1150.49 -8444.49 5978.13 -2242.5 8292.65 -3220.29 5927.76
3934.51 1180.07 6984.37 -4246.9 8637.11 -7779.1 -2090.76 -972.16 5081.91 3757.52 5867.48 -1364.98 -2253.01 -2317.04 7538.48
2567.1 4127.5 -7301.12 3633.44 4506.93 3899.58 5279.76 -9575.27 -5381.81 111.13 -4644.28 8031.24 8316.73 -2657.04 -4902.05 758.7
-3894.01 1298.77 -2278.06 -9084.62 679.99

Sources:

https://rosettacode.org/wiki/Binary_search

https://www.geeksforgeeks.org/quick-sort/

https://www.youtube.com/watch?v=mB5HXBb_HY8&t=707s

https://www.youtube.com/watch?v=7h1s2SojIRw&t=611s

https://www.youtube.com/watch?v=uEUXGcc2VXM