ASD
Niestacjonarne
Egzamin

	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
T:	11	5	7	2	15	4	6	12	87	15	6	45	41	18	17	39	24	64	36	22	17	91	92	96	97

Zadanie 0.

Weryfikacja ćwiczeń.

Proszę zaproponować algorytm znajdujący amplitudę wartości w tablicy
i wykonać go dla poniższej tablicy.

11	5	7	2	15	4	6	12	87	15	6	45	41	18	17	39	24	64	36	22	17	91	92	96	97

Rozwiązanie (C++):

```
 int amplituda(int *tab, int n){
    if(n==0)return 0;
    int max = tab[0];
    int min = tab[0];
    for(int i=0;i<n;i++){
        if(tab[i]>max)max=tab[i];
        if(tab[i]<min)min=tab[i];
    }
    return max - min;
}
```

Wynik: 95

Zadania 0 - 55 p.

Zadanie 1. 20 p.
Proszę wykonać CountingSort dla poniższej tablicy.

1	1	7	2	5	4	6	2	7	1	6	4	4	8	7	3	2	6	6	2	7	1	9	6	7

Rozwiązanie (Python):

```
def counting_sort(tab):
    if len(tab) == 0:
        return tab
    tmin = min(tab)
    tmax = max(tab)
    NC = max - min + 1
    counts = [0] * NC
    for i in range(len(tab)):
        counts[tab[i] - tmin] += 1
    for i in range(1, NC):
        counts[i] += counts[i - 1]
    wynik = [0] * len(tab)
    for i in range(len(tab)):
        wynik[counts[tab[i] - tmin] - 1] = tab[i]
        counts[tab[i] - tmin] -= 1
    return wynik
```

Tabele:

counts:
[4, 8, 9, 12, 13, 18, 23, 24, 25]
wynik:
[1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 9]

Zadanie 2. 20 p.
Proszę wykonać SelectionSort dla poniższej tablicy.

11	5	7	2	15	4	6	12	87	36	22	17	91	92	96	97

Rozwiązanie.
c++

```
void selection_sort(int *tab, int n){
    if(n==0)return;
    for(int j=0;j<n-1;j++){
        // znajduje najmniejszy element spośród podanych
        int min_index = j;
        for(int i=j+1;i<n;i++){
            if(tab[i]<tab[min_index]) min_index = i;
        }
        // zamienia miejscami najmniejszy element z elementem z indeksem j
        int x = tab[j];
        tab[j] = tab[min_index];
        tab[min_index] = x;
    }
}
```

python

```
def selection_sort(tab):
    if len(tab) == 0:
        return tab
    for i in range(len(tab)):
        min_index = i
        for j in range(i, len(tab)):
            if tab[j] < tab[min_index]:
                min_index = j
        tab[i], tab[min_index] = tab[min_index], tab[i]
    return tab
```

Kolejne tabele (min_index, tab):

```
0  [11, 5, 7, 2, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
3  [2, 5, 7, 11, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
5  [2, 4, 7, 11, 15, 5, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
5  [2, 4, 5, 11, 15, 7, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
```

```
6  [2, 4, 5, 6, 15, 7, 11, 12, 87, 36, 22, 17, 91, 92, 96, 97]
5  [2, 4, 5, 6, 7, 15, 11, 12, 87, 36, 22, 17, 91, 92, 96, 97]
6  [2, 4, 5, 6, 7, 11, 15, 12, 87, 36, 22, 17, 91, 92, 96, 97]
7  [2, 4, 5, 6, 7, 11, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
7  [2, 4, 5, 6, 7, 11, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
11 [2, 4, 5, 6, 7, 11, 12, 15, 17, 36, 22, 87, 91, 92, 96, 97]
10 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
10 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
11 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
12 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
13 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
14 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
15 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
```

Zadanie 3. 25 p.

Proszę wykonać InsertionSort dla poniższej tablicy.

11	5	7	2	15	4	6	12	87	36	22	17	91	92	96	97

Rozwiązanie (numer powtórzenia, wygląd tabeli):

```
1  [5, 11, 7, 2, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
2  [5, 7, 11, 2, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
3  [2, 5, 7, 11, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
4  [2, 5, 7, 11, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
5  [2, 4, 5, 7, 11, 15, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
6  [2, 4, 5, 6, 7, 11, 15, 12, 87, 36, 22, 17, 91, 92, 96, 97]
7  [2, 4, 5, 6, 7, 11, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
8  [2, 4, 5, 6, 7, 11, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
9  [2, 4, 5, 6, 7, 11, 12, 15, 36, 87, 22, 17, 91, 92, 96, 97]
10 [2, 4, 5, 6, 7, 11, 12, 15, 22, 36, 87, 17, 91, 92, 96, 97]
11 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
12 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
13 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
14 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
15 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
```

Zadania 56 - 75 p.

Zadanie 4. 10 p.

Proszę wykonać BubbleSort dla poniższej tablicy.

11	5	7	2	15	4	6	12	87	36	22	17	91	92	96	97

Rozwiązanie:

```
def bubble_sort(tab):
    if len(tab) == 0:
        return tab
    swapped = True
    for i in range(len(tab) - 1):
        swapped = False
        for j in range(len(tab) - i - 1):
            if tab[j] > tab[j + 1]:
                tab[j], tab[j + 1] = tab[j + 1], tab[j]
                swapped = True
            print(i,j, tab)
        if not swapped:
            break
    return tab
```

Kolejne tabele po iteracjach (i, j, tab):

```
0 0  [5, 11, 7, 2, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 1  [5, 7, 11, 2, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 2  [5, 7, 2, 11, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 3  [5, 7, 2, 11, 15, 4, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 4  [5, 7, 2, 11, 4, 15, 6, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 5  [5, 7, 2, 11, 4, 6, 15, 12, 87, 36, 22, 17, 91, 92, 96, 97]
0 6  [5, 7, 2, 11, 4, 6, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
0 7  [5, 7, 2, 11, 4, 6, 12, 15, 87, 36, 22, 17, 91, 92, 96, 97]
0 8  [5, 7, 2, 11, 4, 6, 12, 15, 36, 87, 22, 17, 91, 92, 96, 97]
0 9  [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 87, 17, 91, 92, 96, 97]
0 10 [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
0 11 [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
0 12 [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
0 13 [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
0 14 [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 0  [5, 7, 2, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 1  [5, 2, 7, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 2  [5, 2, 7, 11, 4, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 3  [5, 2, 7, 4, 11, 6, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 4  [5, 2, 7, 4, 6, 11, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 5  [5, 2, 7, 4, 6, 11, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 6  [5, 2, 7, 4, 6, 11, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 7  [5, 2, 7, 4, 6, 11, 12, 15, 36, 22, 17, 87, 91, 92, 96, 97]
1 8  [5, 2, 7, 4, 6, 11, 12, 15, 22, 36, 17, 87, 91, 92, 96, 97]
1 9  [5, 2, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
1 10 [5, 2, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
1 11 [5, 2, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
1 12 [5, 2, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
1 13 [5, 2, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 0  [2, 5, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 1  [2, 5, 7, 4, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 2  [2, 5, 4, 7, 6, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 3  [2, 5, 4, 6, 7, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 4  [2, 5, 4, 6, 7, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 5  [2, 5, 4, 6, 7, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 6  [2, 5, 4, 6, 7, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 7  [2, 5, 4, 6, 7, 11, 12, 15, 22, 17, 36, 87, 91, 92, 96, 97]
2 8  [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
2 9  [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
2 10 [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
2 11 [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
2 12 [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 0  [2, 5, 4, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 1  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 2  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 3  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 4  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 5  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
```

```
3 6  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 7  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 8  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 9  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 10 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
3 11 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 0  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 1  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 2  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 3  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 4  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 5  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 6  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 7  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 8  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 9  [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
4 10 [2, 4, 5, 6, 7, 11, 12, 15, 17, 22, 36, 87, 91, 92, 96, 97]
```

Zadanie 5. 15 p.

Proszę wykonać RadixSort dla poniższej tablicy.

111	25	477	2	15	414	316	212	87	915	56	345	341	118	517	539	324	64	436	122	717	191	592	496	697

Rozwiązanie (python):

```
def radix_sort(tab):
    if len(tab) == 0:
        return tab
    exp = 1
       # powtarzamy tak długo dopóki pozycja cyfry znaczącej jest mniejsza od
maksymalnej
    while exp <= max(tab):
        buckets = [[] for i in range(10)]
        for i in range(len(tab)):
            # dzielimy elementy na kontenery
            buckets[(tab[i] // exp) % 10].append(tab[i])
        tab = []


        for bucket in buckets:
            # łączę elementy z kontenerem
            tab.extend(bucket)
        exp *= 10
    return tab
```

Wygląd tabeli buckets dla kolejnych miejsc znaczących:

Jedności:

```
0: []
1: [111, 341, 191]
```

```
2: [2, 212, 122, 592]
3: []
4: [414, 324, 64]
5: [25, 15, 915, 345]
6: [316, 56, 436, 496]
7: [477, 87, 517, 717, 697]
8: [118]
9: [539]
Tabela: [111, 341, 191, 2, 212, 122, 592, 414, 324, 64, 25, 15, 915, 345, 316, 56,
436, 496, 477, 87, 517, 717, 697, 118, 539]

Dziesiątki:
0: [2]
1: [111, 212, 414, 15, 915, 316, 517, 717, 118]
2: [122, 324, 25]
3: [436, 539]
4: [341, 345]
5: [56]
6: [64]
7: [477]
8: [87]
9: [191, 592, 496, 697]
Tabela: [2, 111, 212, 414, 15, 915, 316, 517, 717, 118, 122, 324, 25, 436, 539,
341, 345, 56, 64, 477, 87, 191, 592, 496, 697]

Setki:
0: [2, 15, 25, 56, 64, 87]
1: [111, 118, 122, 191]
2: [212]
3: [316, 324, 341, 345]
4: [414, 436, 477, 496]
5: [517, 539, 592]
6: [697]
7: [717]
8: []
9: [915]
Tabela: [2, 15, 25, 56, 64, 87, 111, 118, 122, 191, 212, 316, 324, 341, 345, 414,
436, 477, 496, 517, 539, 592, 697, 717, 915]
```

Zadania 76 - 100 p.

Zadanie 6. 15 p.
Proszę wykonać HeapSort dla poniższej tablicy.

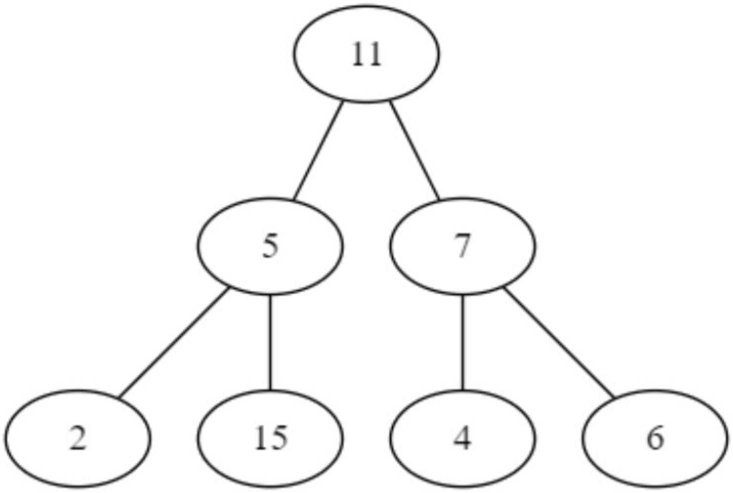
11	5	7	2	15	4	6

Rozwiązanie (python):

```
def heapsort(tab):
    if len(tab) == 0:
        return tab

    # tworzymy strukturę drzewa binarnego z elementów tablicy
    # każdemu elementowi na i-tym indeksie poza 0 (pień główny) odpowiada jeden i
dokładnie jeden element pnia
    # indeks pnia jest równy (i-1)//2 (dzielenie całkowite przez 2)

    output = []
    while len(tab) > 1:
        for i in range(1, len(tab)):
            # oblicz poziom pnia
            k = len(tab) - i
            r = (k-1)//2
            # jeśli liść jest większy od pnia, zamień je miejscami
            if tab[k] > tab[r]:
                tab[k], tab[r] = tab[r], tab[k]

        output.append(tab[0])
        tab = tab[1:]

        print(output, tab)
    output += tab
    return output
```

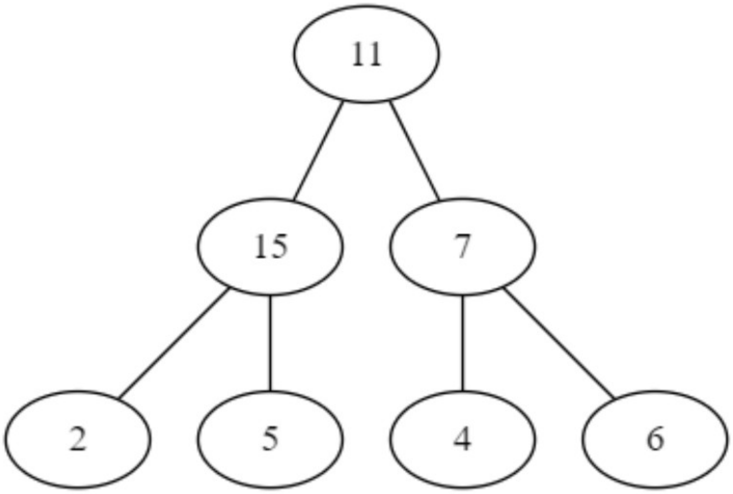
Wygląd tabeli w kolejnych krokach (output, tab):

Początek:



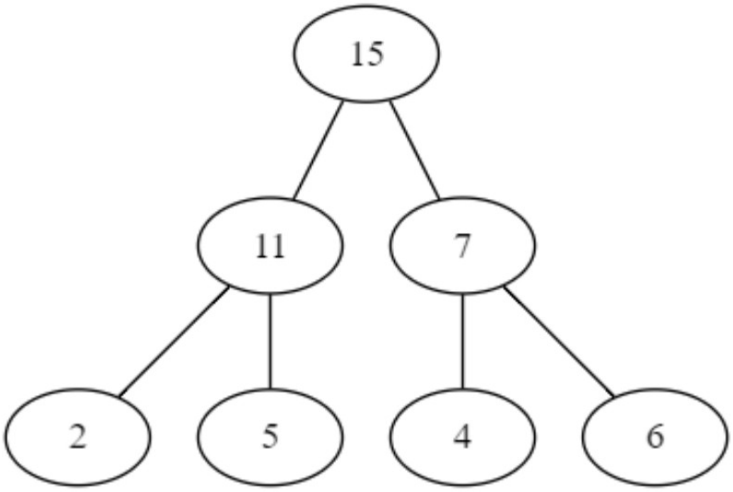
`Wygląd tabeli: [11, 5, 7, 2, 15, 4, 6]`

Wykonujemy zamianę 15 z 5:



```
Wygląd tabeli: [11, 5, 7, 15, 2, 4, 6]
```

Wykonujemy zamianę 15 z 11:

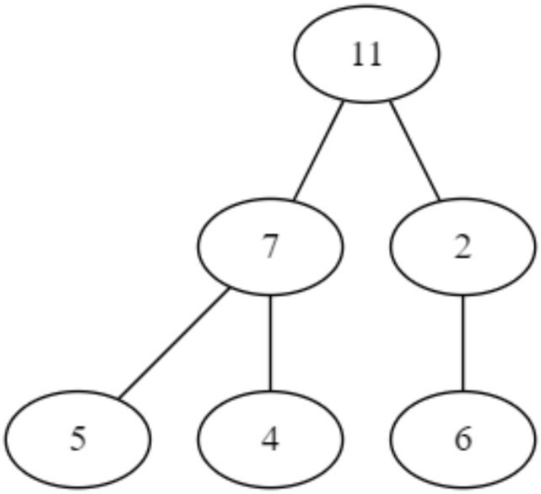


```
Wygląd tabeli: [15, 5, 7, 11, 2, 4, 6]
```

Wyodrębniamy 15 z drzewa. Wygląd tablicy (ostateczna, dane):

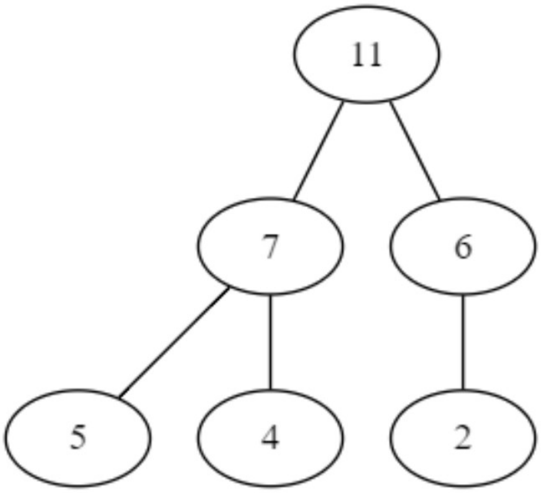
```
[15] [11, 7, 2, 5, 4, 6]
```

```
Kolejny krok. Tworzymy drzewo:
```



```
Wygląd tabeli: [11, 7, 2, 5, 4, 6]
```

```
Zamieniamy 6 z 2:
```

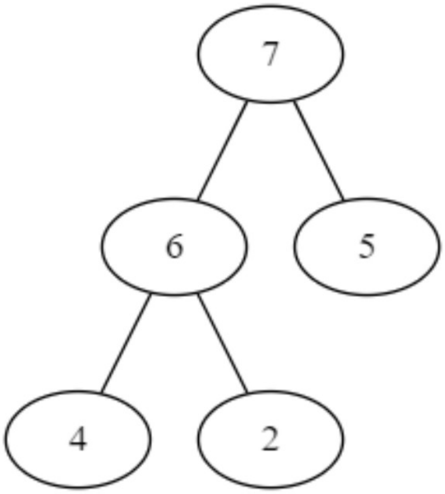


Wygląd tabeli: [11, 7, 6, 5, 4, 2]

Wyodrębniamy 11. Zostaje nam:

[15, 11] [7, 6, 5, 4, 2]

Kolejny krok. Drzewo:

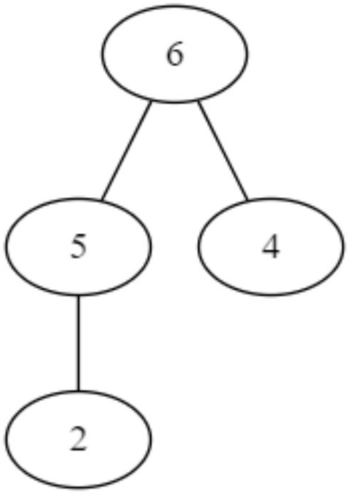


Wygląd tabeli: [7, 6, 5, 4, 2]

Brak podmian. Wyodrębniamy 7. Tablice:

[15, 11, 7] [6, 5, 4, 2]

Kolejny krok:

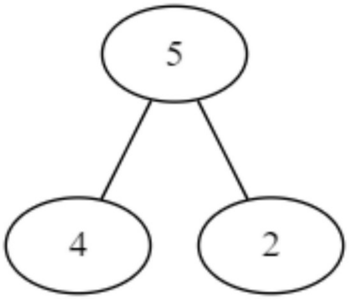


Wygląd tabeli: [6, 5, 4, 2]

Brak zmian. Wyodrębniamy 6.

[15, 11, 7, 6] [5, 4, 2]

Kolejny krok:

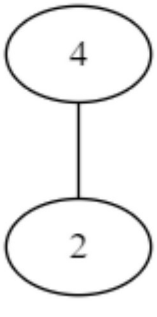


Wygląd tabeli: [5, 4, 2]

Brak zmian. Wyodrębniamy 5.

[15, 11, 7, 6, 5] [4, 2]

Kolejny krok:



Wygląd tabeli: [4, 2]

Wyodrębniamy 4.

[15, 11, 7, 6, 5, 4] [2]

Zostaje nam tylko tabela: [2]. Przyłączamy ją do tablicy. Wygląd tablicy:

[15, 11, 7, 6, 5, 4, 2]

Tablica po odwróceniu:

[2, 4, 5, 6, 7, 11, 15]

Zadanie 7. 15 p.
Proszę wykonać Partitioning dla poniższej tablicy.

11	5	7	2	15	4	6	12	87	15	6	45	41	18	17	39	24	64	36	22	17	91	92	96	97

Rozwiązanie:

```
def partitioning(tab, left, right):
    # wybieramy środkowy element jako nasz pivot
    pivot = tab[(left + right) // 2]
    # powtarzamy tak długo dopóki indeks lewej strony jest mniejszy bądź równy od
prawej
    while left <= right:
        # zawiększ indeks lewej strony tak długo, dopóki element odpowiadajacy
        # indeksowi lewej strony jest mniejszy od pivotu
        while tab[left] < pivot:
            left += 1
        # zrób analogicznie z prawej strony, tym razem zmienjszając tak długo
dopóki jest większy od pivotu
        while tab[right] > pivot:
            right -= 1

        # kiedy masz już indeksy lewej i prawej strony, zamien elementy miejscami,
a następnie zwiększ
        # indeks lewej strony o 1 a indeks prawej strony zmniejsz o 1.
        # Jeśli indeks lewej strony jest większy lub równy indeksowi prawej strony,
zakończ działanie.

        print(left, right, tab)

        if left <= right:
            tab[left], tab[right] = tab[right], tab[left]
            left += 1
            right -= 1
    return left
```

Dane w kolejnych iteracjach (indeks left, indeks right, tablica):

pivot: 41
zamiana: [11, 5, 7, 2, 15, 4, 6, 12, **87**, 15, 6, 45, 41, 18, 17, 39, 24, 64, 36, 22, **17**, 91, 92, 96, 97]
po zamianie: [11, 5, 7, 2, 15, 4, 6, 12, **17**, 15, 6, 45, 41, 18, 17, 39, 24, 64, 36, 22, **87**, 91, 92, 96, 97]

zamiana: [11, 5, 7, 2, 15, 4, 6, 12, 17, 15, 6, **45**, 41, 18, 17, 39, 24, 64, 36, **22**, 87, 91, 92, 96, 97]
po zamianie: [11, 5, 7, 2, 15, 4, 6, 12, 17, 15, 6, **22**, 41, 18, 17, 39, 24, 64, 36, **45**, 87, 91, 92, 96, 97]

zamiana: [11, 5, 7, 2, 15, 4, 6, 12, 17, 15, 6, 22, **41**, 18, 17, 39, 24, 64, **36**, 45, 87, 91, 92, 96, 97]
po zamianie: [11, 5, 7, 2, 15, 4, 6, 12, 17, 15, 6, 22, **36**, 18, 17, 39, 24, 64, **41**, 45, 87, 91, 92, 96, 97]

Postać końcowa. Na zielono jest zaznaczony pivot.
[11, 5, 7, 2, 15, 4, 6, 12, 17, 15, 6, 22, 36, 18, 17, 39, 24, 64, 41, **45**, 87, 91, 92, 96, 97]