Median filter

Реализовал 3 метода подсчета медианного фильтра

`easy` - O(R^2)

`medium` - O(R)

`hard` - O(1)

In [1]:

```
import ctypes
import struct
import sys
```

In [2]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
```

In [3]:

```
SMALL_SIZE = 12
MEDIUM_SIZE = 15
BIGGER_SIZE = 18

plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=MEDIUM_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y label
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=MEDIUM_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

In [4]:

```
from CPP_MedianFilter.python_packges.PyMedianFilter import PyMedianFilte
```

In [5]:

```
def show_img(img, ax=plt):
    if len(img.shape) == 2:
        ax.imshow(img, cmap='gray')
    else:
        ax.imshow(img)
```

In [6]:

```
!ls ./CPP_MedianFilter/resources/inputs/
```

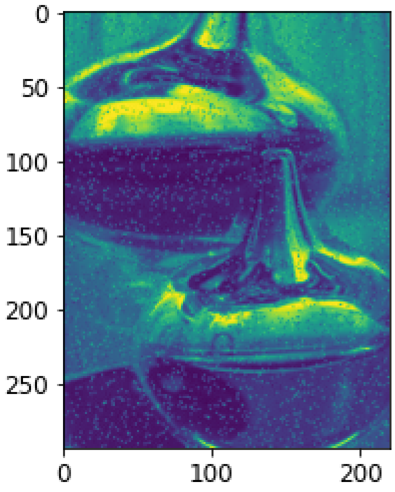
```
image0.jpeg  image2.jpeg  image4.jpeg  image6.jpeg  image8.jpeg
image1.jpeg  image3.jpeg  image5.jpeg  image7.jpeg  image9.jpeg
```

In [7]:

```
images = [mpimg.imread(f'./CPP_MedianFilter/resources/inputs/image{i}.jp
for i, img in enumerate(images):
    if len(img.shape) == 2:
        images[i] = img[..., np.newaxis]
show_img(images[6])
for img in images:
    print(img.shape)
```

```
(257, 196, 3)
(225, 225, 3)
```

```
(224, 225, 3)
(226, 223, 3)
(225, 225, 3)
(288, 512, 3)
(294, 220, 1)
(359, 478, 3)
(450, 800, 3)
(661, 1000, 3)
```

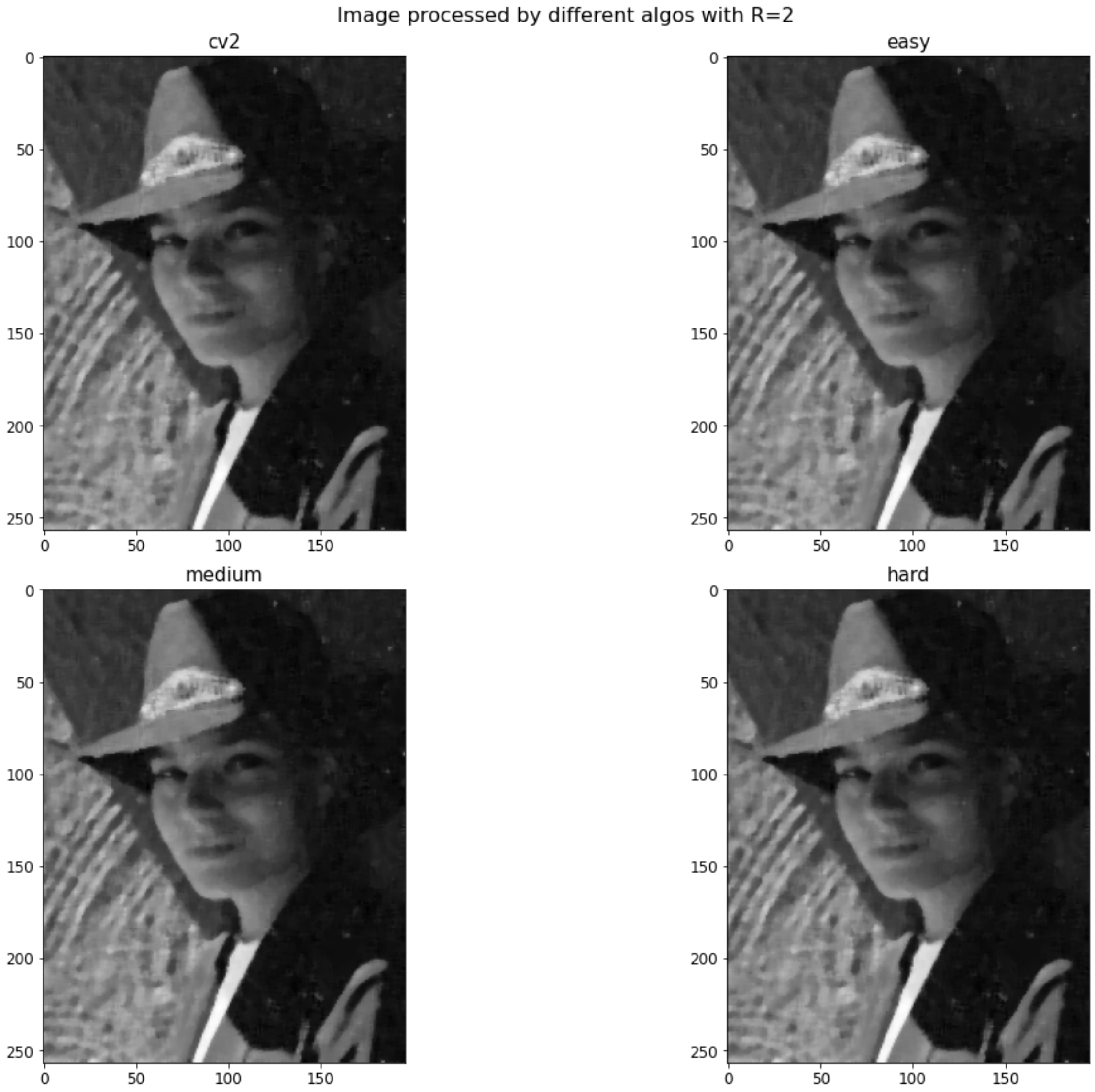


In [8]:

```
algos = {
    'cv2': lambda img, r : cv2.medianBlur(img, r * 2 + 1),
    'easy': PyMedianFilter().process_easy,
    'medium': PyMedianFilter().process_medium,
    'hard': PyMedianFilter().process_hard,
}
```

In [9]:

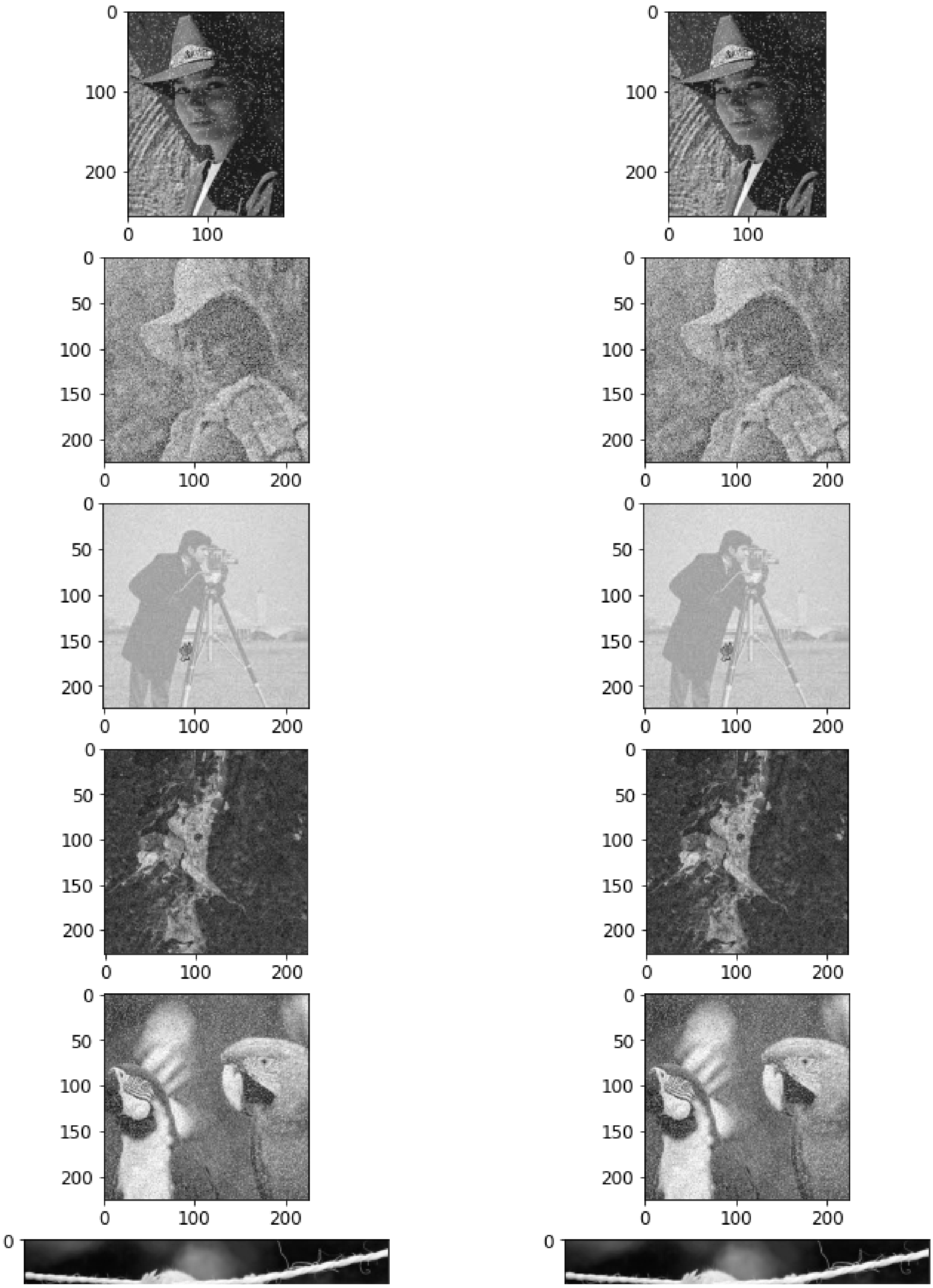
```
def test_algos(img):
    axes_x = 2
    R = 2
    axes_y = len(algos) // 2 + len(algos) % 2
    fig, axes = plt.subplots(axes_y, axes_x, constrained_layout=True)
    fig.suptitle(f'Image processed by different algos with R={R}', fonts
    fig.set_figwidth(15)
    fig.set_figheight(12)

    res_cv = np.array(algos['cv2'](img, R))

    for i, (algo_name, algo) in enumerate(algos.items()):
        algo_ax = axes[i // 2, i % 2]
        res = np.array(algo(img, R))
        assert((res == res_cv).all()) #  main_check
        show_img(res, ax=algo_ax)
        algo_ax.set_title(algo_name)
test_algos(images[0][:,:,0])
```

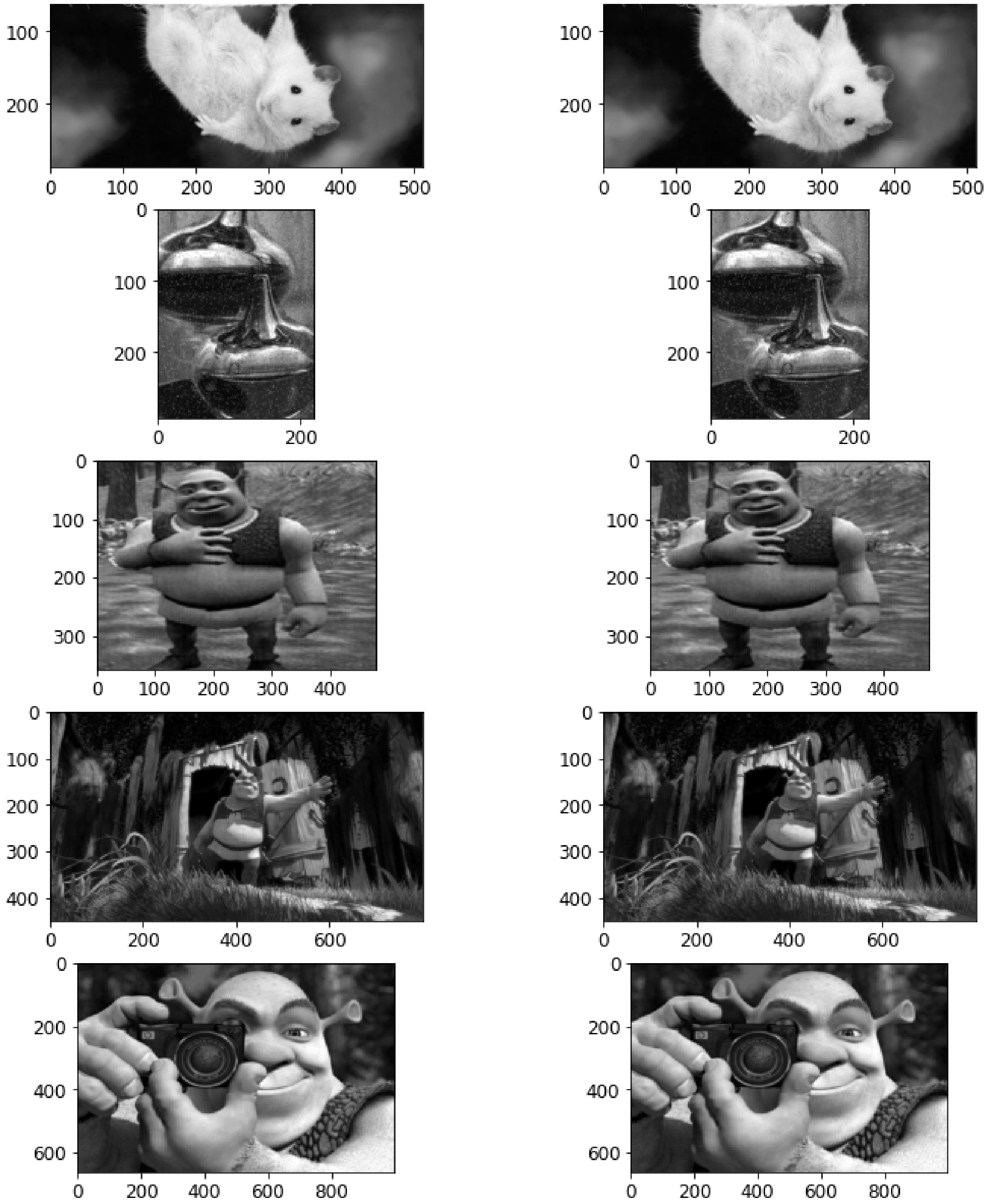


In [10]:

```
import time

def show_images(imgs, algo_name='easy', R=0):
    axes_x = 2
    axes_y = len(imgs)
    fig, axes = plt.subplots(axes_y, axes_x)
#     fig.tight_layout(pad=3.0)
    fig.suptitle(f'Images processed by median filter[cv/{algo_name}] wit
    fig.set_figwidth(12)
    fig.set_figheight(30)


    time_cv = []
    time_my = []
    for i, img in enumerate(imgs):
        size_mp = img.shape[0] * img.shape[1] / 1000000
        start_time = time.time()
        res_cv = np.array(algos['cv2'](img[:,:,0], R))
        time_cv.append((time.time() - start_time) * 1000 / size_mp)

        start_time = time.time()
        res_my = np.array(algos[algo_name](img[:,:,0], R))
        time_my.append((time.time() - start_time) * 1000 / size_mp)

        show_img(res_cv, ax=axes[i,0])
        show_img(res_my, ax=axes[i,1])
    return time_cv, time_my
time_cv, time_my = show_images(images, algo_name='easy')
# print(time_cv)
# print(time_my)
```

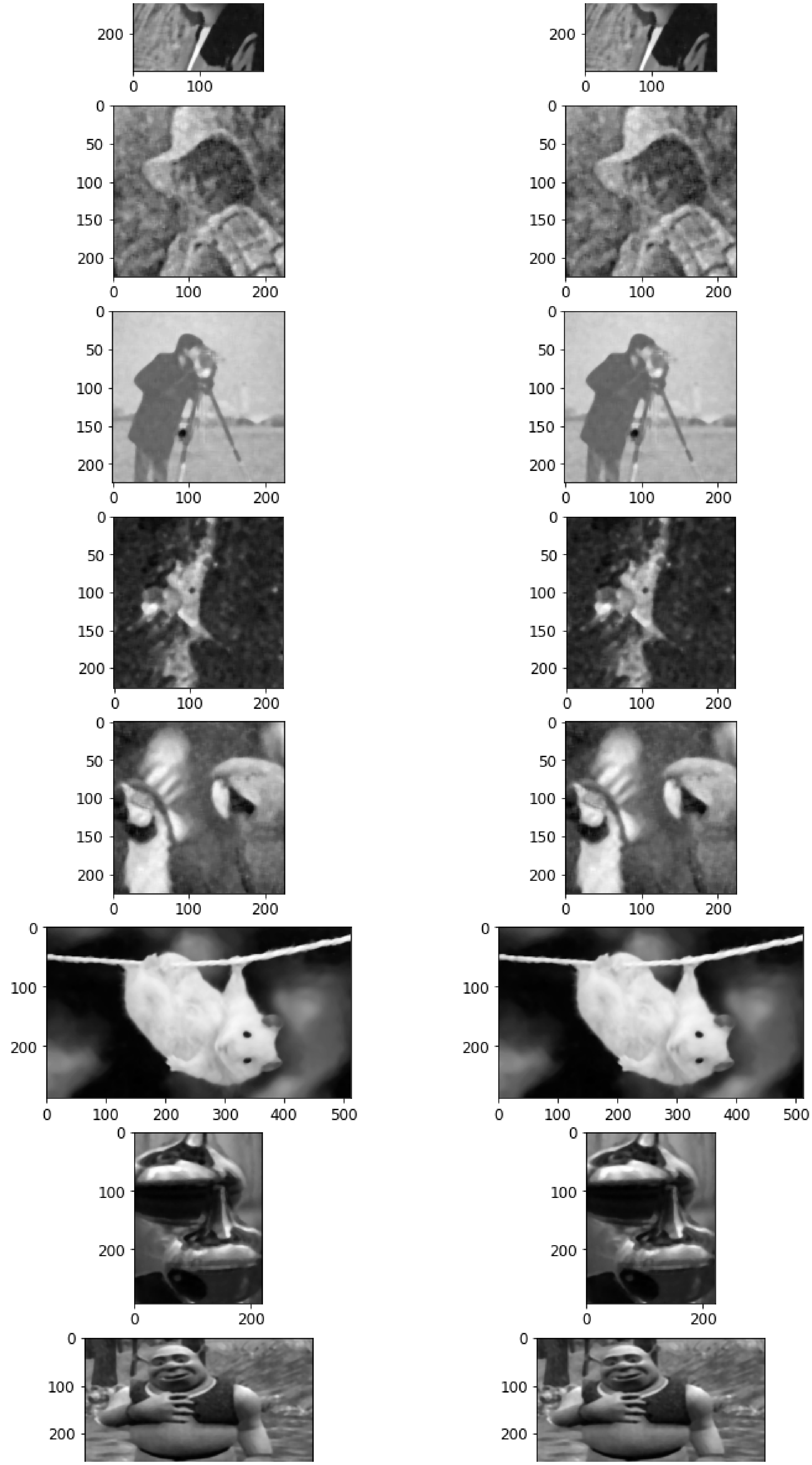
Images processed by median filter[cv/easy] with R=0

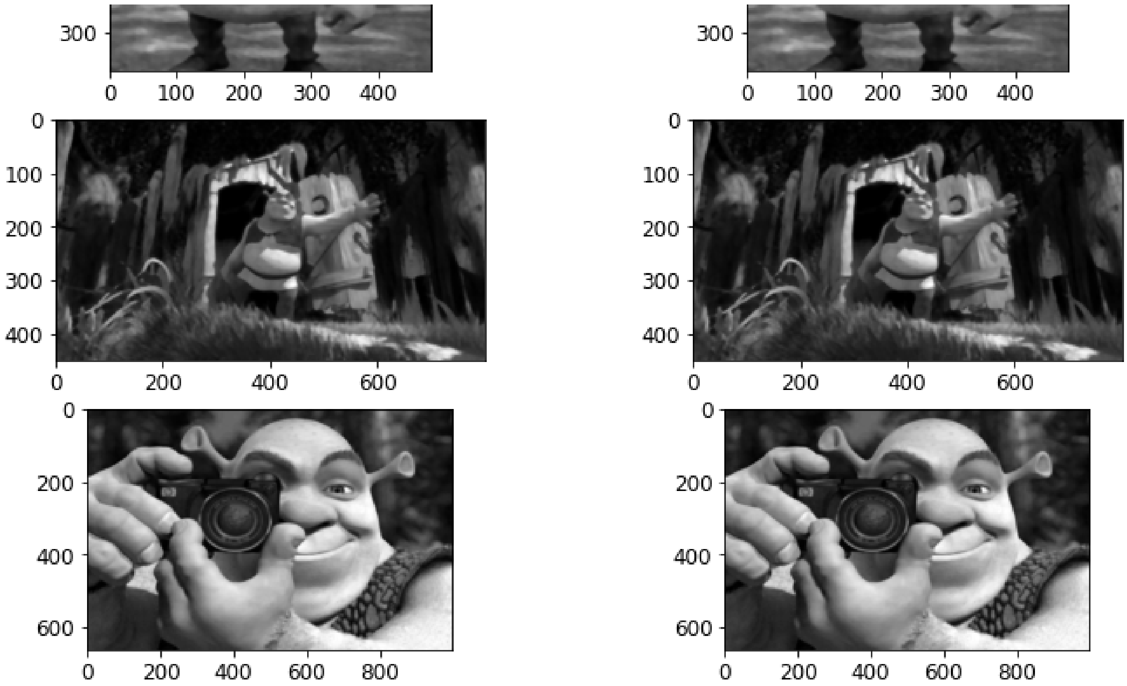




In [11]:
```
time_cv, time_my = show_images(images, algo_name='medium', R=3)
```

Images processed by median filter[cv/medium] with R=3

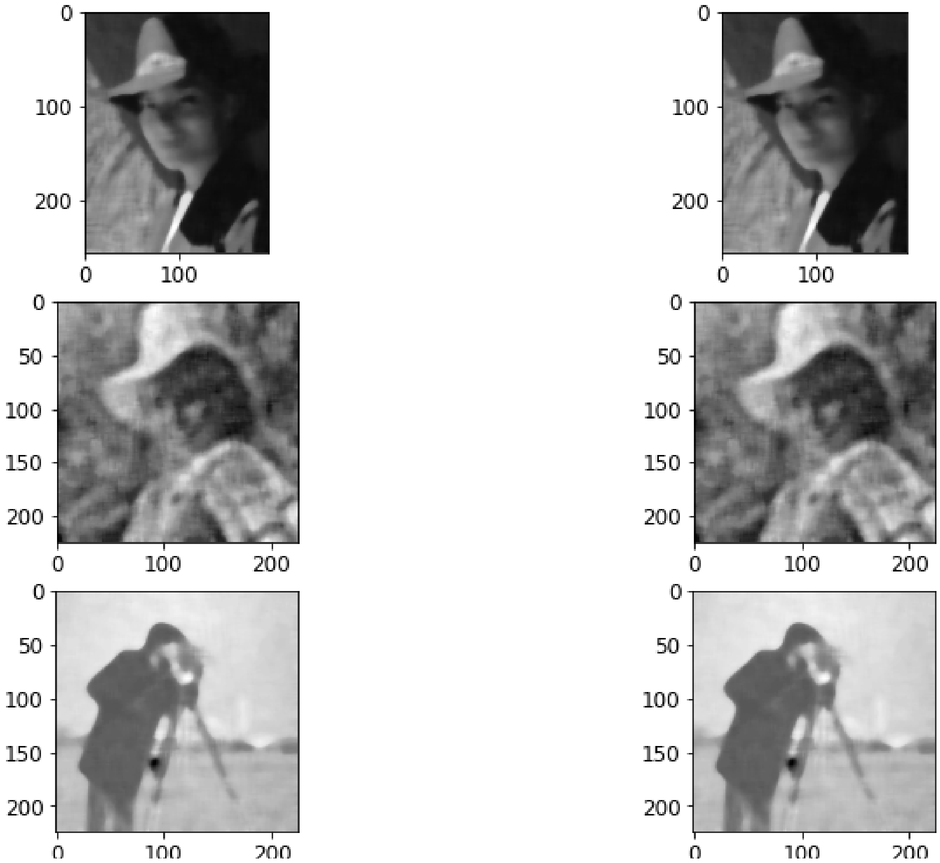

200
0
100
200
0
100
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
100
200
0
100
200
300
400
500
0
100
200
0
200
0
100
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
100
200
0
100
200
300
400
500
0
100
200
0
200
0
100
200

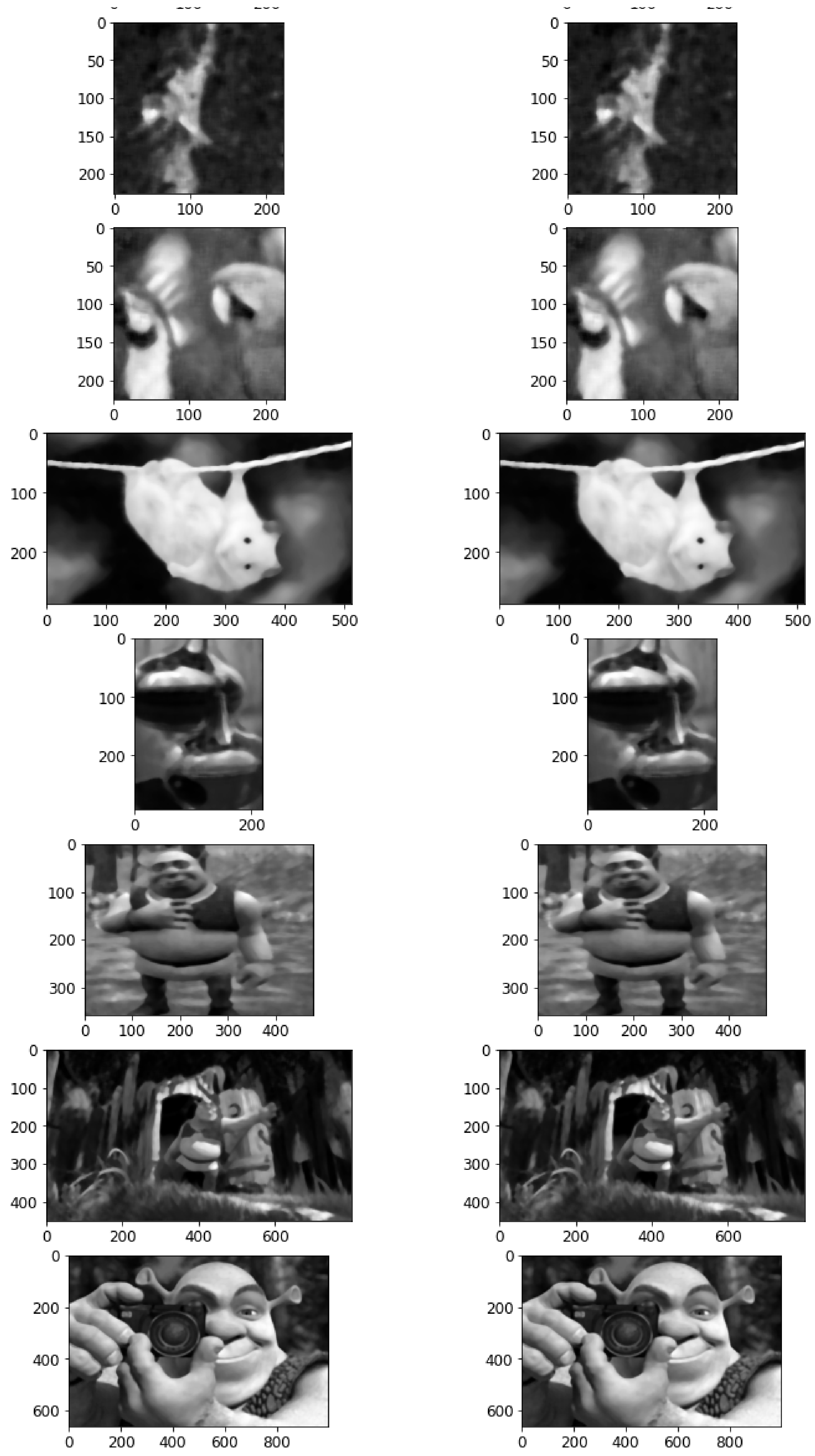


In [12]:

```
time_cv, time_my = show_images(images, algo_name='medium', R=5)
```

Images processed by median filter[cv/medium] with R=5



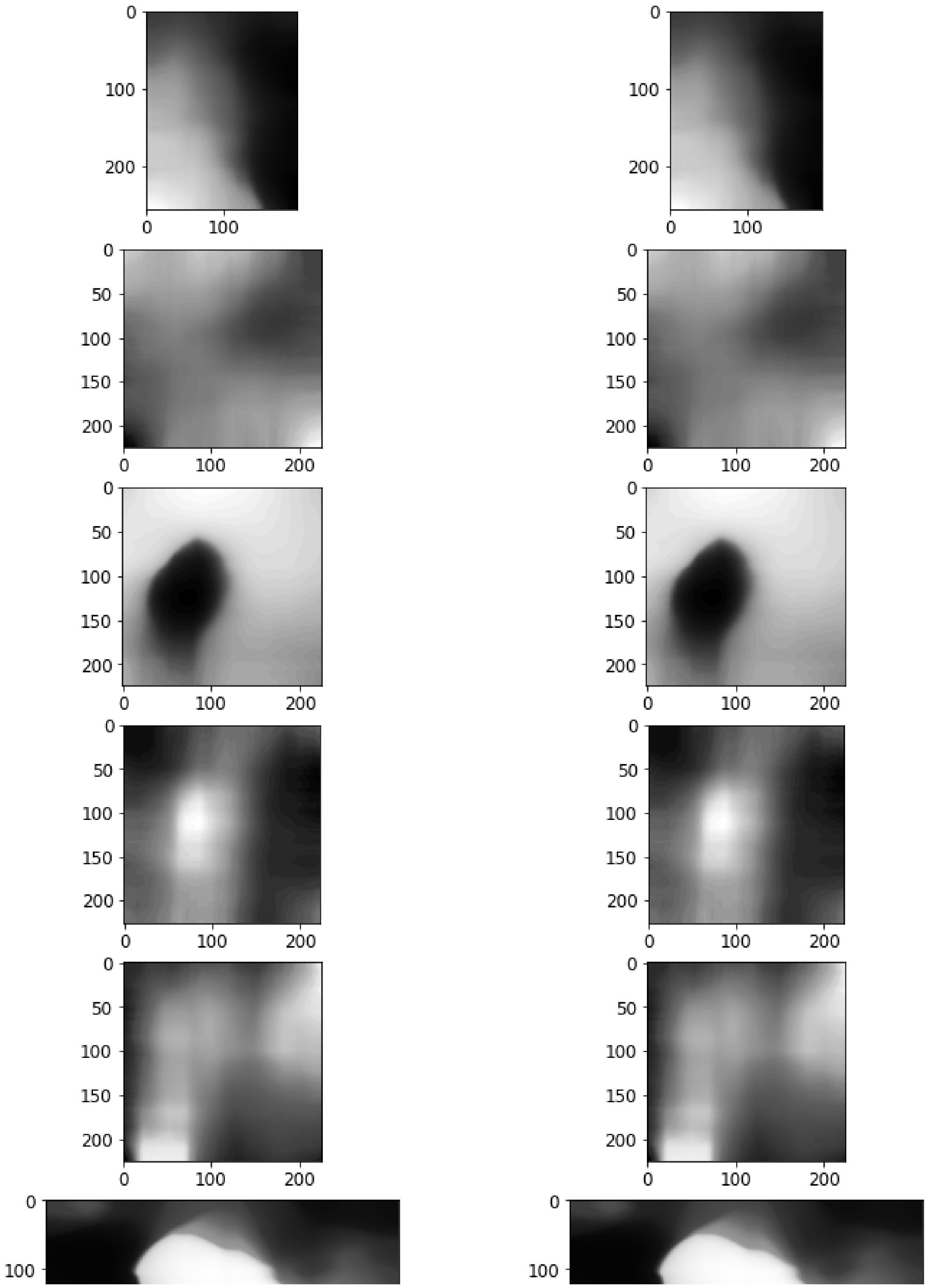


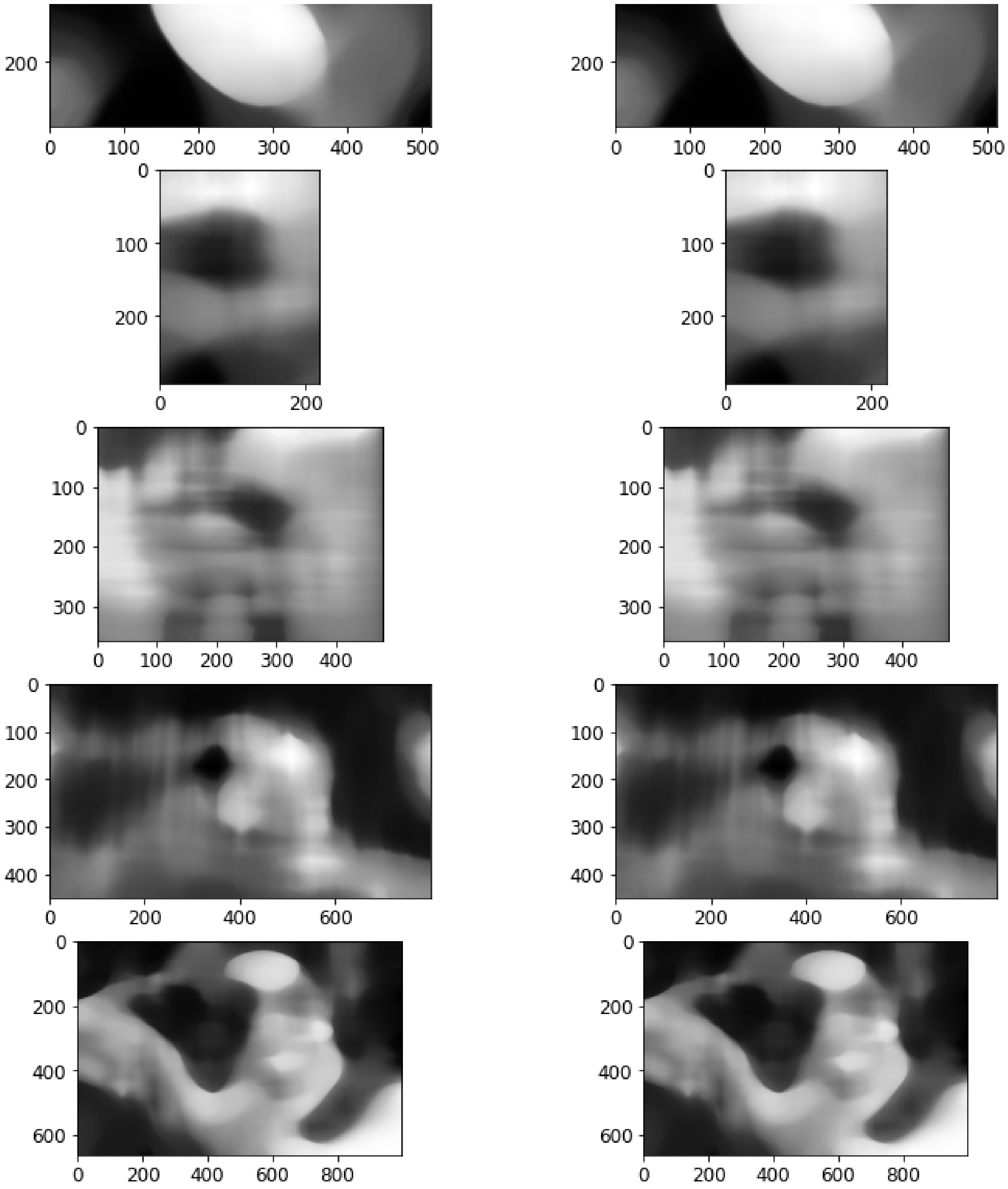
0 50 100 150 200
0 100 200
0 50 100 150 200
0 100 200
0 50 100 150 200
0 100 200
0 50 100 150 200
0 100 200
0 100 200
0 100 200 300 400 500
0 100 200
0 100 200 300 400 500
0 100 200
0 200
0 100 200
0 200
0 100 200 300
0 100 200 300 400
0 100 200 300
0 100 200 300 400
0 100 200 300 400
0 200 400 600
0 100 200 300 400
0 200 400 600
0 200 400 600
0 200 400 600 800
0 200 400 600
0 200 400 600 800

In [13]:

```
time_cv, time_my = show_images(images, algo_name='medium', R=50)
```

Images processed by median filter[cv/medium] with R=50





In [14]:

```
def process(imgs, algo=algos['easy'], R=2):
    time_my = []
    for i, img in enumerate(imgs):
        size_mp = img.shape[0] * img.shape[1] / 1000000
        start_time = time.time()
        res_my = np.array(algo(img[:,:,0], R))
        time_my.append((time.time() - start_time) * 1000 / size_mp)

    return time_my
process(images, algo=algos['medium'], R=5)
```

Out[14]:

```
[594.7826948630121,
 615.5131775655864,
 521.7109407697405,
 650.8990892214767,
 559.8751115210262,
 452.11183735065987,
 517.6573088674834,
 451.71041141082793,
 462.34528223673504,
 425.5403729321917]
```

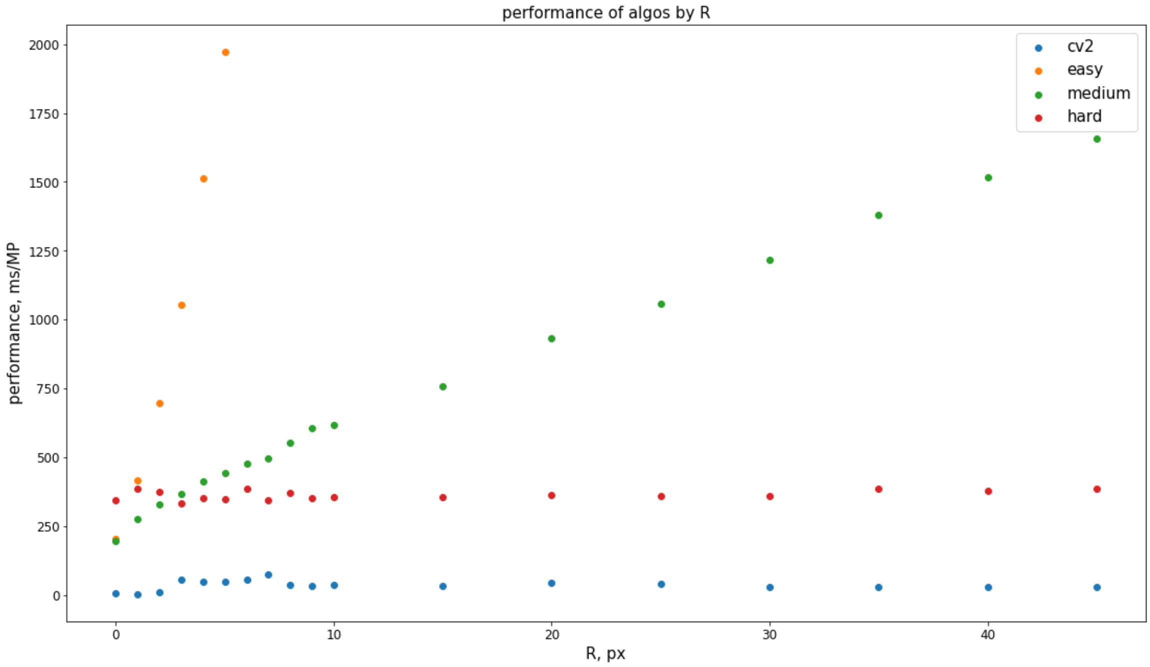
In [15]:

```python
R_range = list(range(10)) + list(range(10, 50, 5))
print(f'Range {R_range}')
algos_res = {
    'cv2': [],
    'easy': [],
    'medium': [],
    'hard': [],
}
for algo_name in ['cv2', 'easy', 'medium', 'hard']:
    for R in R_range:
        if algo_name == 'easy' and R > 5: # too slow on big R
            continue
        time_algo = process(images, algo=algos[algo_name], R=R)
        result = np.mean(time_algo)
        algos_res[algo_name].append(result)
        print(f'{algo_name} mean: {result}ms/MP for {R}')
```

```
Range [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45]
cv2 mean: 5.696571259789928ms/MP for 0
cv2 mean: 3.8693357732136824ms/MP for 1
cv2 mean: 9.752665608387511ms/MP for 2
cv2 mean: 56.00864607005335ms/MP for 3
cv2 mean: 49.20579097515633ms/MP for 4
cv2 mean: 49.82582667870029ms/MP for 5
cv2 mean: 58.30101447315046ms/MP for 6
cv2 mean: 75.38626897645574ms/MP for 7
cv2 mean: 38.0502543813889ms/MP for 8
cv2 mean: 32.482009896796804ms/MP for 9
cv2 mean: 38.58420539242384ms/MP for 10
cv2 mean: 31.79855973937078ms/MP for 15
cv2 mean: 43.79325132552022ms/MP for 20
cv2 mean: 42.3433717980336ms/MP for 25
cv2 mean: 29.846730890999403ms/MP for 30
cv2 mean: 30.832253478536796ms/MP for 35
cv2 mean: 30.913563966878247ms/MP for 40
cv2 mean: 30.0264793398404ms/MP for 45
easy mean: 202.4602671918721ms/MP for 0
easy mean: 416.5561896757801ms/MP for 1
easy mean: 698.2975377896408ms/MP for 2
easy mean: 1053.6759477084188ms/MP for 3
easy mean: 1514.2105002948042ms/MP for 4
easy mean: 1972.0323486018065ms/MP for 5
medium mean: 197.4796508509518ms/MP for 0
medium mean: 278.0560506228687ms/MP for 1
medium mean: 329.96493367629716ms/MP for 2
medium mean: 367.61587120244855ms/MP for 3
medium mean: 412.4584059222093ms/MP for 4
medium mean: 444.78112183666224ms/MP for 5
medium mean: 479.0677966059837ms/MP for 6
medium mean: 494.75348076114216ms/MP for 7
medium mean: 553.8907206825513ms/MP for 8
medium mean: 604.7215821466095ms/MP for 9
medium mean: 618.8899096742459ms/MP for 10
medium mean: 759.6358049867431ms/MP for 15
medium mean: 932.5782572202461ms/MP for 20
medium mean: 1056.5300416832968ms/MP for 25
medium mean: 1217.4126092392128ms/MP for 30
medium mean: 1380.1202374913032ms/MP for 35
medium mean: 1515.6586873177414ms/MP for 40
medium mean: 1656.7654923922764ms/MP for 45
hard mean: 346.10336676826586ms/MP for 0
hard mean: 388.0010752806531ms/MP for 1
hard mean: 373.57425780524045ms/MP for 2
hard mean: 332.6330339700345ms/MP for 3
hard mean: 351.86877050767964ms/MP for 4
hard mean: 349.84248270043025ms/MP for 5
hard mean: 385.5193713204155ms/MP for 6
hard mean: 343.70147257449815ms/MP for 7
hard mean: 372.42073705317745ms/MP for 8
hard mean: 350.5949373660911ms/MP for 9
hard mean: 357.0432794186514ms/MP for 10
hard mean: 356.7049824172917ms/MP for 15
hard mean: 363.16232079730315ms/MP for 20
hard mean: 361.27848816891094ms/MP for 25
hard mean: 359.89321802367147ms/MP for 30
hard mean: 385.64053117068806ms/MP for 35
hard mean: 379.89180103262504ms/MP for 40
hard mean: 387.5874884262375ms/MP for 45
```

In [16]:

```python
def show_results(algo_names=['cv2', 'easy','medium', 'hard']):
    fig, axes = plt.subplots(1, 1)
    fig.set_size_inches(18.5, 10.5, forward=True)
    for algo_name in algo_names:
        axes.scatter(x=R_range[:len(algos_res[algo_name])], y=algos_res[
        axes.legend()
        axes.set_title('performance of algos by R')
        axes.set_ylabel('performance, ms/MP')
        axes.set_xlabel('R, px')

show_results()
```



In [17]:

```python
print(np.array(algos_res['medium']) < np.array(algos_res['hard']))
th_combined = np.sum(np.array(algos_res['medium']) < np.array(algos_res[
print(f'Наш порог для комбинированного алгоритма: {th_combined}')
```

```
[ True  True  True False False False False False False False False False
 False False False False False False]
Наш порог для комбинированного алгоритма: 3
```

Вывод

По графику видно, что самый простой алгоритм вообще не выгодно использовать

При радиусе окна < 4 есть смысл использовать алгоритм за линию.

Заметим, что фактическая асимтотика похожа на теоритическую.

Жаль, что без векторизации не добиться такой же скорости, что и в cv2(

In [18]:

```
def combined_algo(img, r):
    # the easiest case :)
    if r == 0:
        return img
    if r <= th_combined:
        return algos['medium'](img, r)
    else:
        return algos['hard'](img, r)
algos['combined'] = combined_algo
```

In [19]:

```
R_range = list(range(10)) + list(range(10, 50, 5))
print(f'Range {R_range}')
algos_res = {
    'cv2': [],
    'easy': [],
    'medium': [],
    'hard': [],
    'combined': []
}
for algo_name in ['cv2', 'easy', 'medium', 'hard', 'combined']:
    for R in R_range:
        if algo_name == 'easy' and R > 5: # too slow on big R
            continue
        time_algo = process(images, algo=algos[algo_name], R=R)
        result = np.mean(time_algo)
        algos_res[algo_name].append(result)
        print(f'{algo_name} mean: {result}ms/MP for {R}')
```

```
Range [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45]
cv2 mean: 2.344999741720504ms/MP for 0
cv2 mean: 9.327651404341326ms/MP for 1
cv2 mean: 18.066698772663987ms/MP for 2
cv2 mean: 47.91838113085187ms/MP for 3
cv2 mean: 53.59806038402144ms/MP for 4
cv2 mean: 53.23756502805753ms/MP for 5
cv2 mean: 61.00370060755124ms/MP for 6
cv2 mean: 61.17312666359802ms/MP for 7
cv2 mean: 31.377363699105548ms/MP for 8
cv2 mean: 34.22089390806654ms/MP for 9
cv2 mean: 39.695909595405645ms/MP for 10
cv2 mean: 31.908773817411987ms/MP for 15
cv2 mean: 32.0294206849904ms/MP for 20
cv2 mean: 32.33638838157441ms/MP for 25
cv2 mean: 33.874684810105926ms/MP for 30
cv2 mean: 31.194783444311916ms/MP for 35
cv2 mean: 36.704742882721604ms/MP for 40
cv2 mean: 29.00946893222268ms/MP for 45
easy mean: 156.92232479922808ms/MP for 0
easy mean: 389.41547983565613ms/MP for 1
easy mean: 717.171160857947 5ms/MP for 2
easy mean: 1062.6998809847016ms/MP for 3
easy mean: 1442.736507073212 8ms/MP for 4
easy mean: 1918.7692742078827ms/MP for 5
medium mean: 199.284456414721 4ms/MP for 0
medium mean: 275.6710953984024ms/MP for 1
medium mean: 331.2824821621008ms/MP for 2
medium mean: 346.4369914791551ms/MP for 3
medium mean: 409.51507326657367ms/MP for 4
medium mean: 439.66156706796016ms/MP for 5
medium mean: 488.1611842559879ms/MP for 6
medium mean: 502.49371834403775ms/MP for 7
medium mean: 619.6687373503567ms/MP for 8
medium mean: 580.3206344389707ms/MP for 9
medium mean: 652.3484866351562ms/MP for 10
medium mean: 911.2107706180119ms/MP for 15
medium mean: 1012.93136399733ms/MP for 20
medium mean: 1183.03148628619ms/MP for 25
medium mean: 1193.4386191327965ms/MP for 30
medium mean: 1385.4053663779832ms/MP for 35
medium mean: 1571.6194408208453ms/MP for 40
medium mean: 1703.4993421733889ms/MP for 45
hard mean: 344.3774265958433ms/MP for 0
hard mean: 343.5475367163873ms/MP for 1
hard mean: 353.3540350286412ms/MP for 2
hard mean: 343.9983444574176ms/MP for 3
hard mean: 351.44595893946337ms/MP for 4
hard mean: 349.9079950635215ms/MP for 5
hard mean: 364.33147048412724ms/MP for 6
hard mean: 365.59286417627544ms/MP for 7
hard mean: 372.63708358407246ms/MP for 8
hard mean: 373.3215848960325ms/MP for 9
hard mean: 373.04619776474243ms/MP for 10
hard mean: 438.61075518576524ms/MP for 15
hard mean: 369.1639192707087ms/MP for 20
hard mean: 359.6576572180057ms/MP for 25
hard mean: 358.5698253002609ms/MP for 30
hard mean: 368.6841089255163ms/MP for 35
hard mean: 377.8146418554141ms/MP for 40
hard mean: 388.4756928879593ms/MP for 45
combined mean: 1.0944762395899013ms/MP for 0
combined mean: 279.5356969640003ms/MP for 1
combined mean: 378.3034933117351ms/MP for 2
combined mean: 363.0417558523999ms/MP for 3
combined mean: 352.13676931380854ms/MP for 4
combined mean: 347.7873031029659ms/MP for 5
combined mean: 363.9710512217871ms/MP for 6
combined mean: 363.4266765465835ms/MP for 7
combined mean: 386.3195956832149ms/MP for 8
combined mean: 362.0902078799678ms/MP for 9
combined mean: 360.622904405151ms/MP for 10
combined mean: 381.85112342366193ms/MP for 15
combined mean: 385.01051938546317ms/MP for 20
combined mean: 368.0637555851953ms/MP for 25
combined mean: 379.3737017736929ms/MP for 30
combined mean: 567.1721870933438ms/MP for 35
combined mean: 371.4636040140425ms/MP for 40
combined mean: 376.2266038235513ms/MP for 45
```

In [20]:

```
test_algos(images[0][:,:,0])
```

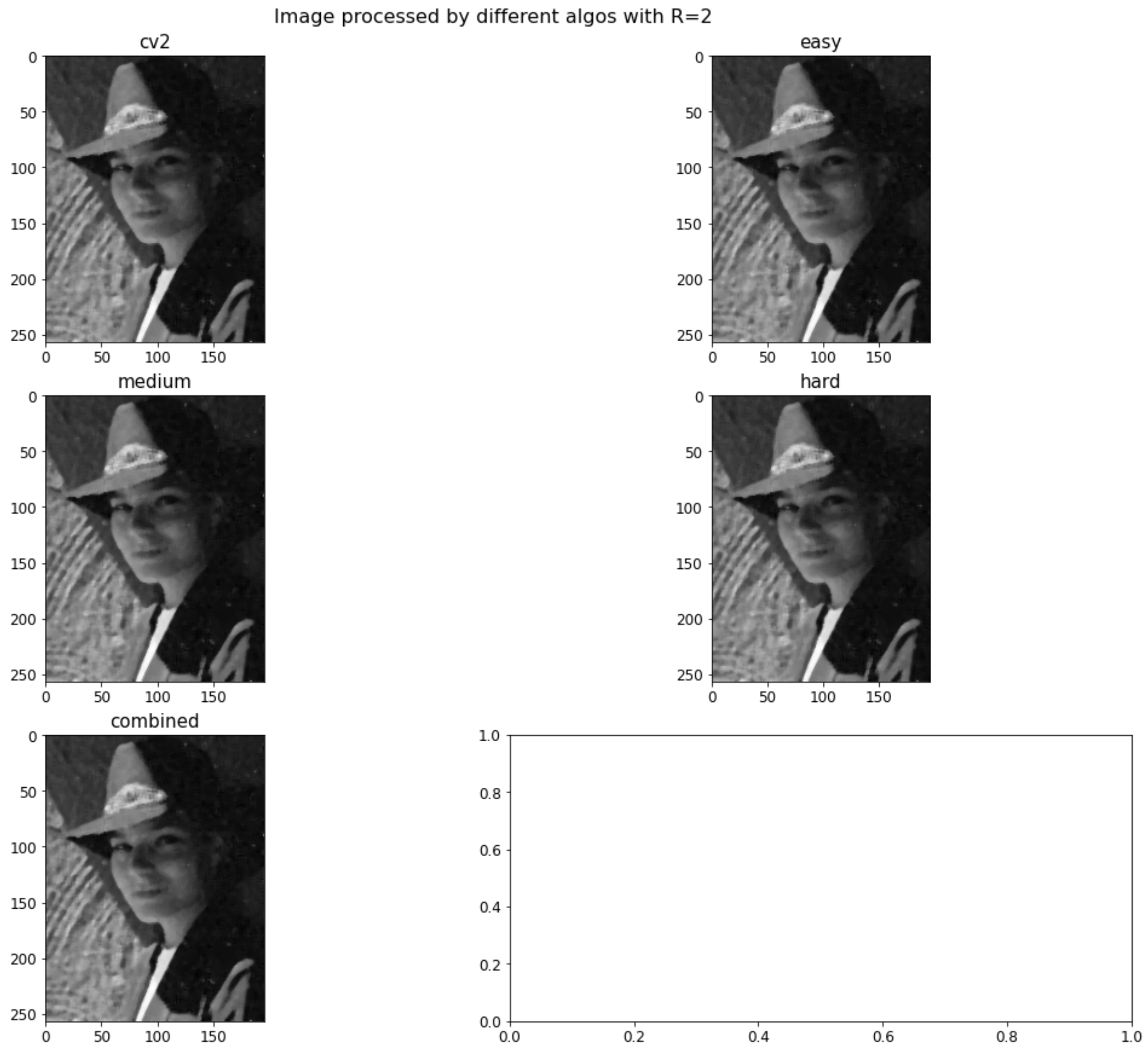


Image processed by different algos with R=2

cv2

easy

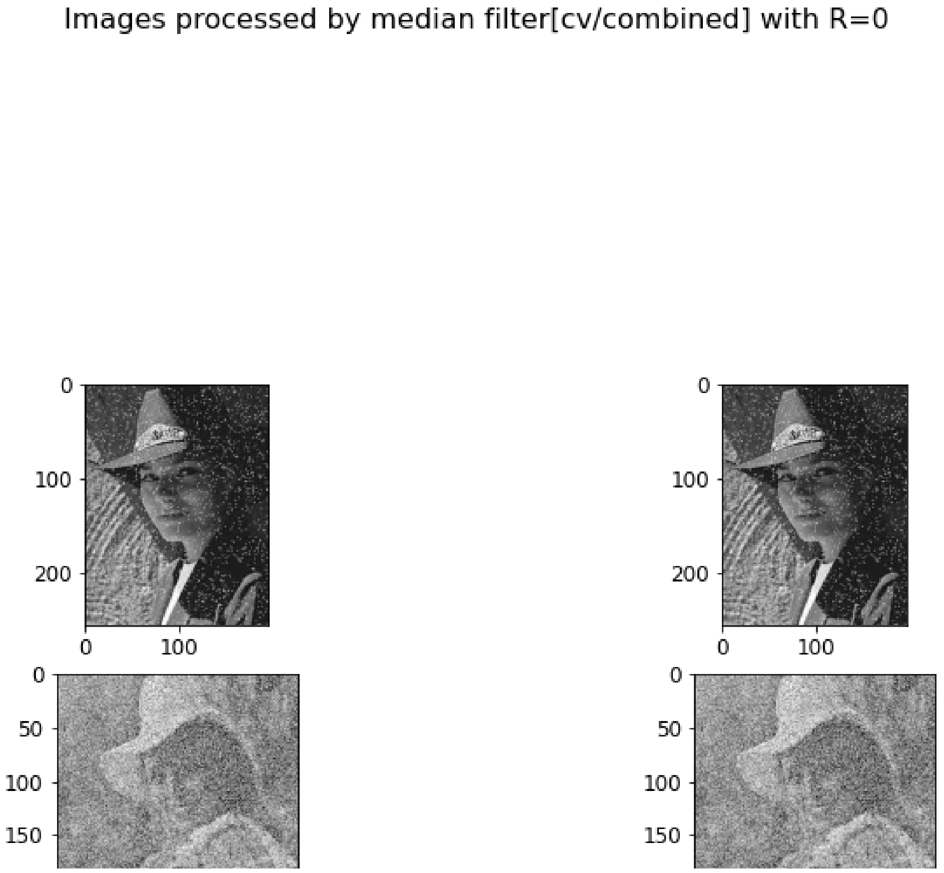
medium

hard

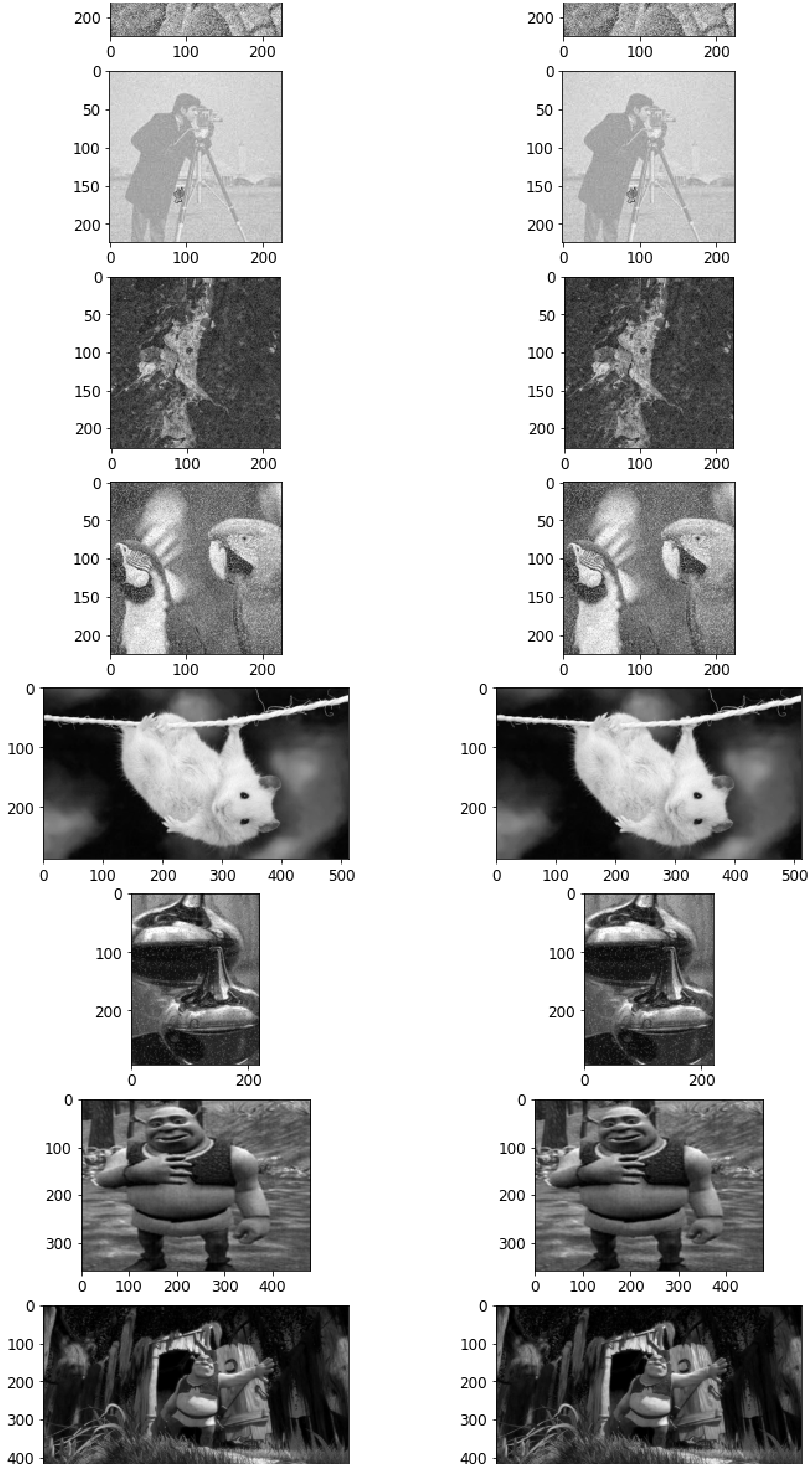
combined

In [21]:

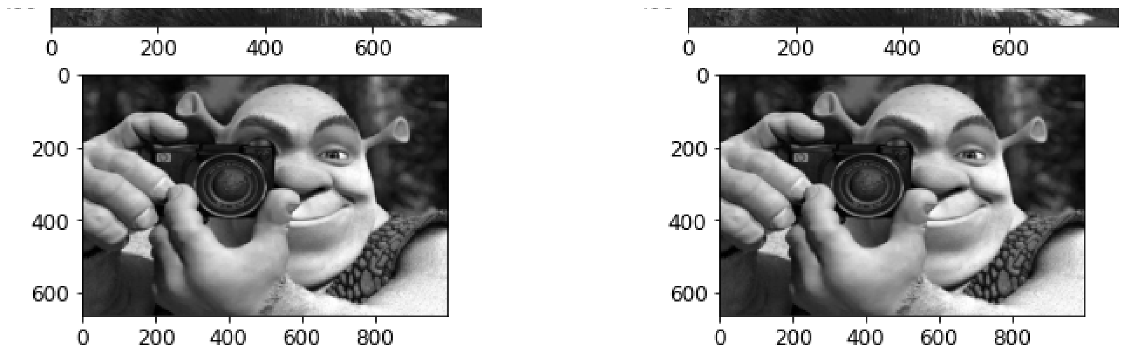
```
time_cv, time_my = show_images(images, algo_name='combined')
```



Images processed by median filter[cv/combined] with R=0

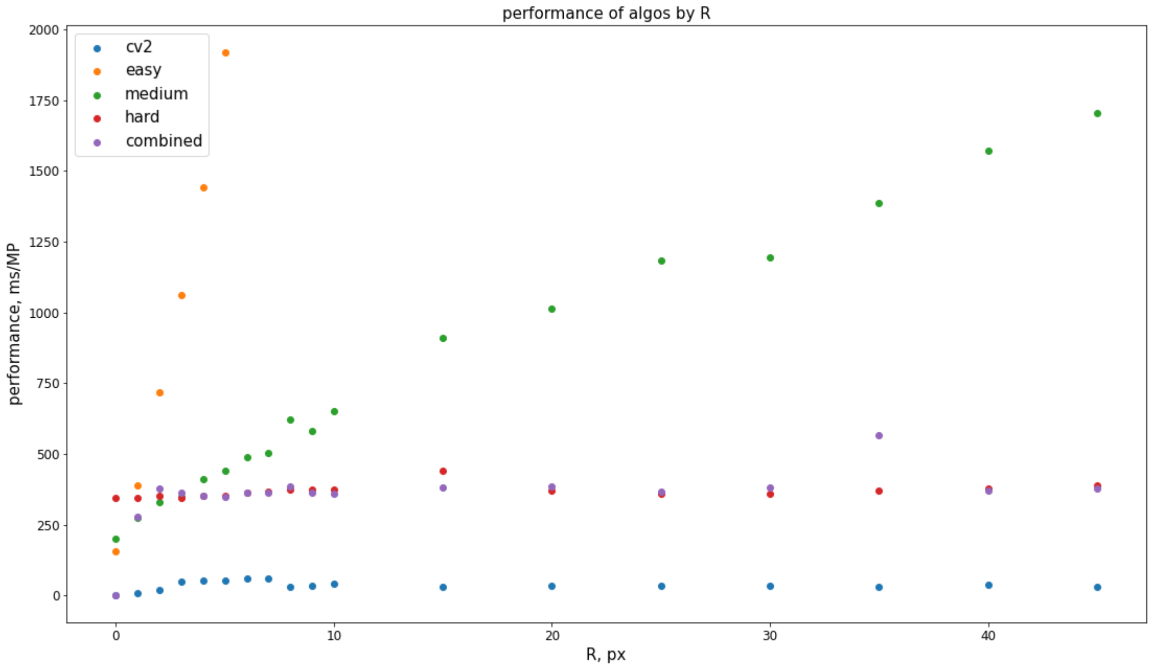


200
0
100
200
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
50
100
150
200
0
100
200
0
100
200
0
100
200
300
400
500
0
100
200
0
100
200
300
400
500
0
100
200
0
200
0
100
200
0
200
0
100
200
300
0
100
200
300
400
0
100
200
300
0
100
200
300
400
0
100
200
300
400
0
100
200
300
400



In [22]:

```
show_results(['cv2', 'easy', 'medium', 'hard', 'combined'])
```



In [23]:

```
def print_img_cpp_vector():
    for i in img[:10,:10,0]:
        s = ','.join(map(str, i))
        print(f'{{{s}}},')
```