In [1]:
```
import numpy as np
import json
import matplotlib.pyplot as plt
from tqdm import tqdm
import random
import subprocess
import time
import os
```

In [3]:
```
with open("packages_npm.txt") as file:
    packages = file.read().strip().split('\n')
```

In [10]:
```
def get_seconds(time):
    min_ind = time.find('m')
    mins = int(time[:min_ind])
    second = float(time[min_ind + 1:-1])
    return mins * 60 + second

def log(file, msg):
    f = open(file, 'a+')
    f.write(msg + '\n')
    f.close()
```

In [11]:
```
rates_gzip = []
rates_brotli = []
times_gzip = []
times_brotli = []
speed_gzip = []
speed_brotli = []
init_sizes = []
all_urls = []
```

```
for i in range(len(packages)):
    with open("package.txt", "w") as file:
        file.write(packages[i])
    #delete the current node_modules directories containing previou
s package
    result = subprocess.run(["rm", "-rf", "node_modules"])
    #install the package and save the names of js scripts
    result = subprocess.run(["bash", "npm_install_packages.sh"])
    result = subprocess.run(["bash", "find_urls_save.sh"])
    with open("urls_for_package.txt") as file:
        urls = file.read().strip().split('\n')
    all_urls.append(urls)

    #concatenate all scripts of that package together to simulate w
eb bundle
    script_concatenated = ""
    for url in all_urls[i]:
        if url == "":
            continue
        if not os.path.exists(url):
            print(i)
            print("DOESN'T EXIST: ", url)
            continue
        with open(url) as file:
            script_concatenated += file.read()

    rates_gzip_compressed = []
    rates_brotli_compressed = []
    times_gzip_compressed = []
    times_brotli_compressed = []
    speed_gzip_compressed = []
    speed_brotli_compressed = []

    with open("example2.txt", "w") as file:
        file.write(script_concatenated)
    size_non_compressed = os.stat("example2.txt").st_size
    init_sizes.append(size_non_compressed)

    # do the gzip compression with different levels
    for level in range(4, 10):
        result = subprocess.run(["bash", "gzip_compress.sh", str(le
vel), "time2.txt",
                                 "example_gzip2.txt.gz", "example2.
txt"])
        with open("time2.txt") as file:
            user_sys = file.read().strip().split('\n')[1:]
        time = get_seconds(user_sys[0].split('\t')[1]) + get_second
s(user_sys[1].split('\t')[1])
        size_gzip_compressed = os.stat("example_gzip2.txt.gz").st_s
ize
        rates_gzip_compressed.append(size_non_compressed / size_gzi
p_compressed)
        times_gzip_compressed.append(time)
        speed_gzip_compressed.append(size_non_compressed / time)
```

```
    # do the brotli compression with different levels
    for level in range(4, 12):
        result = subprocess.run(["bash", "brotli_compress.sh", str(
level), "time2.txt",
                                 "example_brotli2.txt.br", "example
2.txt"])
        with open("time2.txt") as file:
            user_sys = file.read().strip().split('\n')[1:]
        time = get_seconds(user_sys[0].split('\t')[1]) + get_second
s(user_sys[1].split('\t')[1])
        size_br_compressed = os.stat("example_brotli2.txt.br").st_s
ize
        rates_brotli_compressed.append(size_non_compressed / size_b
r_compressed)
        times_brotli_compressed.append(time)
        speed_brotli_compressed.append(size_non_compressed / time)

    rates_gzip.append(rates_gzip_compressed)
    rates_brotli.append(rates_brotli_compressed)
    times_gzip.append(times_gzip_compressed)
    times_brotli.append(times_brotli_compressed)
    speed_gzip.append(speed_gzip_compressed)
    speed_brotli.append(speed_brotli_compressed)

    if i != 0 and i % 100 == 0:
        log("logs3.txt", "rates_gzip: " + str(np.mean(rates_gzip, a
xis=0)))
        log("logs3.txt", "rates_brotli: " + str(np.mean(rates_brotl
i, axis=0)))
        log("logs3.txt", "times_gzip: " + str(np.mean(times_gzip, a
xis=0)))
        log("logs3.txt", "times_brotli: " + str(np.mean(times_brotl
i, axis=0)))
        log("logs3.txt", "speed_gzip: " + str(np.mean(speed_gzip, a
xis=0)))
        log("logs3.txt", "speed_brotli: " + str(np.mean(speed_brotl
i, axis=0)))
```

In [13]:
```
import pandas as pd
frame = pd.DataFrame()
frame["name"] = ["gzip 4", "gzip 5", "gzip 6", "gzip 7", "gzip 8",
"gzip 9",
                 "brotli 4", "brotli 5", "brotli 6", "brotli 7", "b
rotli 8", "brotli 9", "brotli 10", "brotli 11"]

frame["rates"] = np.hstack((np.mean(rates_gzip, axis=0), np.mean(ra
tes_brotli, axis=0)))
frame["savings"] = 1 - 1 / np.hstack((np.mean(rates_gzip, axis=0),
np.mean(rates_brotli, axis=0)))
frame["speed(MB/s)"] = np.hstack((np.mean(speed_gzip, axis=0), np.m
ean(speed_brotli, axis=0))) / 1000000

frame
```

Out[13]:

	name	rates	savings	speed(MB/s)
0	gzip 4	4.612111	0.783180	58.640409
1	gzip 5	4.830213	0.792970	42.773977
2	gzip 6	4.942799	0.797685	29.547876
3	gzip 7	4.966302	0.798643	24.132054
4	gzip 8	4.984292	0.799370	14.945622
5	gzip 9	4.986703	0.799467	12.569951
6	brotli 4	8.207642	0.878162	48.296430
7	brotli 5	8.530252	0.882770	29.497596
8	brotli 6	9.082333	0.889896	22.155465
9	brotli 7	9.498521	0.894720	14.364543
10	brotli 8	9.713480	0.897050	9.417515
11	brotli 9	9.934222	0.899338	6.265608
12	brotli 10	11.089514	0.909825	1.230184
13	brotli 11	11.378584	0.912116	0.571473

In [25]:
```
print("non compressed size range {}MB-{}MB".format(np.min(init_size
s) / 1000000, np.max(init_sizes)/ 1000000))
```

```
non compressed size range 34.465761MB-81.676873MB
```