6.4.8 More Efficiency Practice

For each of the following, assuming the list L is of length N, indicate the big-O of the function.

Checkpoint 1

```
def f(L):
    r = [ ]
    for i in range(len(L)):
        M = L[i:]
        r.append(M)
    return r
```

- **O(1)**
- **O(N)**
- **O(NlogN)**
- **O($N^{0.5}$)**
- **O(logN)**
- **O(2^N)**
- **O(N^2)**

Submit

Checkpoint 2

```
def f(L):
    M = copy.copy(L)
    s = 0
    while len(M) > 0:
        s += M.pop()
    return s
```

- **O(N^2)**

- ○ **O(1)**
- ○ **O(NlogN)**
- ○ **O(N)**
- ○ **O(logN)**
- ○ **O(2^N)**
- ○ **O($N^{0.5}$)**

Submit

Checkpoint 3

```
def f(L):
    M = copy.copy(L)
    s = 0
    while len(M) > 0:
        s += M.pop(0)
    return s
```

- ○ **O(logN)**
- ○ **O(N^2)**
- ○ **O(N)**
- ○ **O(1)**
- ○ **O(2^N)**
- ○ **O(NlogN)**
- ○ **O($N^{0.5}$)**

Submit

Checkpoint 4

```
def f(L):
    M = [ ]
    sign = +1
    for v in L:
        M.append(sign * v)
        sign = -sign
    return sorted(M)
```

- ○ **$O(N^{0.5})$**
- ○ **O(logN)**
- ○ **O(NlogN)**
- ○ **$O(2^N)$**
- ○ **$O(N^2)$**
- ○ **O(N)**
- ○ **O(1)**

Submit

Checkpoint 5

```
def f(L):
    n = len(L)
    s = 0
    for i in range(0, n, n//4):
        s += L[i]
    return s
```

- ○ **O(N)**
- ○ **$O(N^2)$**
- ○ **$O(N^{0.5})$**
- ○ **$O(2^N)$**
- ○ **O(1)**

- ○ **O(NlogN)**
- ○ **O(logN)**

Submit

Checkpoint 6

```
def f(L):
    M = copy.copy(L)
    n = len(M)
    for i in range(n):
        for j in range(i+1, n):
            if (M[i] + M[j]) % 2 == 0:
                M[i], M[j] = M[j], M[i]
    return M
```

- ○ **O($N^{0.5}$)**
- ○ **O(logN)**
- ○ **O(NlogN)**
- ○ **O(2^N)**
- ○ **O(N)**
- ○ **O(1)**
- ○ **O(N^2)**

Submit

Checkpoint 7

```
def f(L):
    duplicates = set()
    for i in range(len(L)):
        v = L[i]
        if v in L[:i]:
            duplicates.add(v)
    return duplicates
```

- ○ **O(1)**
- ○ **O(N^2)**
- ○ **O(N)**
- ○ **O(2^N)**
- ○ **O(logN)**
- ○ **O(NlogN)**
- ○ **O($N^{0.5}$)**

Submit

Checkpoint 8

```
def f(L):
    seen = set()
    duplicates = set()
    for v in L:
        if v in seen:
            duplicates.add(v)
        seen.add(v)
    return duplicates
```

- ○ **O(1)**
- ○ **O($N^{0.5}$)**
- ○ **O(logN)**

- ○ **O(2^N)**
- ○ **O(N^2)**
- ○ **O(N)**
- ○ **O(NlogN)**

Submit

Checkpoint 9

```
def f(L):
    M = [ ]
    i = 0
    while True:
        M.append(L[i])
        if i**2 >= len(L):
            return M
        i += 1
```

- ○ **O(2^N)**
- ○ **O(NlogN)**
- ○ **O(1)**
- ○ **O($N^{0.5}$)**
- ○ **O(N)**
- ○ **O(logN)**
- ○ **O(N^2)**

Submit

Checkpoint 10

```
def f(L, x):
    i = len(L)
    while i > 0:
        if L[i] == x:
            return True
        i //= 2
    return False
```

- **O(NlogN)**
- **O(2^N)**
- **O(logN)**
- **O(N)**
- **O($N^{0.5}$)**
- **O(1)**
- **O(N^2)**

Submit

Checkpoint 11

```
def f(L):
    counts = dict()
    for v in L:
        counts[v] = L.count(v)
    return counts
```

- **O(NlogN)**
- **O(N^2)**
- **O($N^{0.5}$)**
- **O(N)**
- **O(1)**
- **O(logN)**

- ○ **$O(2^N)$**

Submit

Checkpoint 12

```
def f(L):
    counts = dict()
    for v in L:
        if v not in counts:
            counts[v] = 0
        counts[v] += 1
    return counts
```

- ○ **O(logN)**
- ○ **O(N)**
- ○ **O(NlogN)**
- ○ **O(1)**
- ○ **$O(N^{0.5})$**
- ○ **$O(2^N)$**
- ○ **$O(N^2)$**

Submit

Checkpoint 13

```
def f(L):
    M = [ ]
    for v in L:
        i = 1
        while i < len(L):
            M.append(L[i])
            i *= 2
    return M
```

- ○ **O(1)**
- ○ **O(NlogN)**
- ○ **O(2^N)**
- ○ **O($N^{0.5}$)**
- ○ **O(N)**
- ○ **O(logN)**
- ○ **O(N^2)**

Submit

Checkpoint 14

```
def f(L):
    n = len(L)
    i = 1
    r = 0
    while math.log2(i) <= n:
        r += L[i%n]
        i += 1
    return r
```

- ○ **O(N)**
- ○ **O(2^N)**
- ○ **O(NlogN)**
- ○ **O(logN)**
- ○ **O($N^{0.5}$)**
- ○ **O(1)**
- ○ **O(N^2)**

Submit

Back to Lessons