Hopscotch

Reasoning

We could use dynamic programming to do this. This would mean that each element in $S[j]$ would be the maximum hopscotch count up to that point. It could be found by getting the maximum of either $S[j-2] + S[j]$ or $S[j-3] + S[j]$.

What: $S[j]$ = maximum hopscotch score including point j

How: $S[j] = \begin{cases} \max(S[j-2] + A[j], S[j-3] + A[j]) \text{ for } j > 3 \\ S[0] + A[1] \text{ for } j = 3 \\ 0 \text{ for } j = 2 \\ A[1] \text{ for } j = 1 \end{cases}$

Where: $\max_i S[i]$

Psudocode

```
def find_max_hopscotch_sum(array A of length n):
    if n < 3:
        return A[1]
    let S be a new empty array of length n
    let S[1] get A[1]
    let S[2] get 0
    let S[3] get S[1] + A[3]
    for index from 4 to n:
        S[index] gets max(S[index - 2], S[index-3]) + A[index] # Moved the
addition outside for simplicity, since it is added in either case
    let maximum get 0
    for index from 0 to n:
        if S[index] > maximum:
            maximum = S[index]
    return maximum
```

Proof

First off for the $n < 3$ cases. When there are less then 3 spaces, you can only gain score on the first space, since your first jump will inevitably take you off the array, causing you to gain only the score from the first tile. For the third space, the maximum score must be the score from the first square and the third square, as that is the only way to reach it. The second square is unreachable, so it has a maximum score of zero, ensuring that it is never picked as a jumping off place in the later squares as $S[1]$ or $S[3]$ will always have higher scores and one of them will be compared to $S[2]$. By this logic, we have the highest possible score for the first 3 elements in the S array. For the rest, if we assume that all elements at $S[j]$ represent correctly the highest score you could get up to element

j in the array, then we know that at an element that is at a position $j > 3$, the jump must have come from either $[j-2]$ or $[j-3]$. Since we know that $S[j-2]$ and $S[j-3]$ contain the maximum possible scores at those points, we can tell that the maximum possible score at j must be the maximum of $S[j-2]$ and $S[j-3]$, plus the score that we gain by landing on j or $A[j]$.

Running Time

Estimate

$\Theta(n)$

Reasoning

```
def find_max_hopscotch_sum(array A of length n):
    if n < 3: # O(1)
        return A[1] # O(1)
    let S be a new empty array of length n # O(1)
    let S[1] get A[1] # O(1)
    let S[2] get 0 # O(1)
    let S[3] get S[1] + A[3] # O(1)
    for index from 3 to n:# O(n)
        S[index] gets max(S[index - 2], S[index-3]) + A[index]  # O(1)
    let maximum get 0 # O(1)
    for index from 0 to n: # O(n)
        if S[index] > maximum: # O(1)
            maximum = S[index] # O(1)
    return maximum # O(1)
```

The running time for this algorithm is determined to be $= \{\begin{matrix} cn + d \text{ if } n > 3 \\ d \text{ if } n < 3 \end{matrix}$ And since $c_1 n \leq cn + d \leq c_2 n$ is true beyond a certain point if $c_1 < c < c_2$, the algorithm runs in $\Theta(n)$ time.