Homework 1

Problem 1: (Asymptotic Growth) All the unstated logs are in base 2.

$n2^n$, nlog(n), n!, n, n^{100}, 2^n, log(n), log(n!), (log(n))!, $2^{2^\wedge n}$.

- g_1=log(n)
- g_2=n
- g_3= log (n!)
- g_4=nlog(n)
- g_5=n^{100}
- g_6=(log(n))!
- g_7=2^n
- g_8= $n2^n$
- g_9= n!
- g_{10}=$2^{2^\wedge n}$

$$g_1 = O(g_2),\ g_2 = O(g_3),\ ...,\ g_9 = O(g_{10})$$

Problem 2: (Recurrences) All the unstated logs are in base 2.

- **(a)** T(n) = 2T(n/2) + n^3 solution by master theorem,
  - a=2, b=2;
    - n^log$_b$a= n, f(n)= n^3
    - If ; f(n) = $\Omega(n^\varepsilon)$ then ε= 3 and $2(n/2)^3 \leq cn^3$ for c = ¼;
      - **T (n) = $\Theta(n^3)$ (case 3)**
- **(b)** T(n) = 7T(n/2) + n^2 solution by master theorem,
  - a=7, b=2;
    - n^log$_b$a= $n^{\log 7}$, f(n)= n^2
    - If ; f(n) = $O(n^{2+\log 3-\varepsilon})$ then ε=log3;
      - **T (n) = $\Theta(n^{\log 7})$ (case 1)**
- **(c)** T(n) = 2T(n/4) + n solution by master theorem,
  - a=2, b=4;
    - n^log$_b$a= n^log$_4$2, f(n)= $n^{1/2}$
    - If ; f(n) = $\Theta(n^{1/2}\log^{k+1} n)$ then k=0;
      - **T (n) = $\Theta(n^{1/2} \log n)$ (case 2)**
- **(d)** T(n)=T(n−1) + n solution by substitution,
  - With an educated guess, T(n-1) is iterates n times and in each iteration it's computational complexity is n;
    - Guess = $O(n^2)$, assume that $T(k) \leq ck^2$ for k<n;
    - Prove; $T(n) \leq cn^2$ solution by induction;
      - T(n)=T(n-1) + n
      - $\leq c(n-1)^2 + n = c(n^2-2n+1) + n = cn^2-(c(2n-1)-n)$;
        - $\leq cn^2$ when, (2n-1)c-n ≥0 or (2c-1)n-c ≥0; c≥1 and n≥1
          - **T (n) = $O(n^2)$**

Problem 3: (Binary- Search Python) All the unstated logs are in base 2.

```
def binarySearch(alist,item):
    first = 0
    last = len(alist)-1
    found = False

    while first<=last and not found:
      midpoint = (first + last)/2
      if alist[midpoint] == item:
        found = True
      else:
        if item < alist[midpoint]:
           last = midpoint-1
        else:
           first = midpoint+1
    return found
```

Figure 1: Iterative binary search

```
def binarySearch(alist,item):
    if len(alist) ==  0:
       return False
    else:
       midpoint = len(alist)/2
       if alist[midpoint] == item:
         return True
       else:
         if item<alist[midpoint]:
            return binarySearch(alist[:midpoint],item)
         else:
            return binarySearch(alist[midpoint+1:],item)
```

Figure 2: Recursive binary search

- **(a)**
  - **(i)**
    - By “midpoint = (first + last)/2” statement, function reduces length of the list to half in each iteration, because of that situation upper bound of the function is **O(logn).**
    - In the worst case (item doesn’t exist inside the list), asymptotic running time of the algorithm is **O(logn).**
  - **(ii)**
    - Recurrence is **T(n/2) + n** because,

```
else:
  if item<alist[midpoint]:
     return binarySearch(alist[:midpoint],item)
  else:
     return binarySearch(alist[midpoint+1:],item)
```

If and else part cause “T(n/2)” and copy opetartion by “alist[:midpoint]” cause to “n”. Solution by master theorem,

- a=1, b=2;
- $n^\wedge \log_b a = n^\wedge \log_1 2$, f(n)= n dominates.
  - **T (n) = O(n) (case 3)**