traitement données de fork selon différentes définitions

In [36]:
```
import matplotlib.pyplot as plt
import numpy as np
```

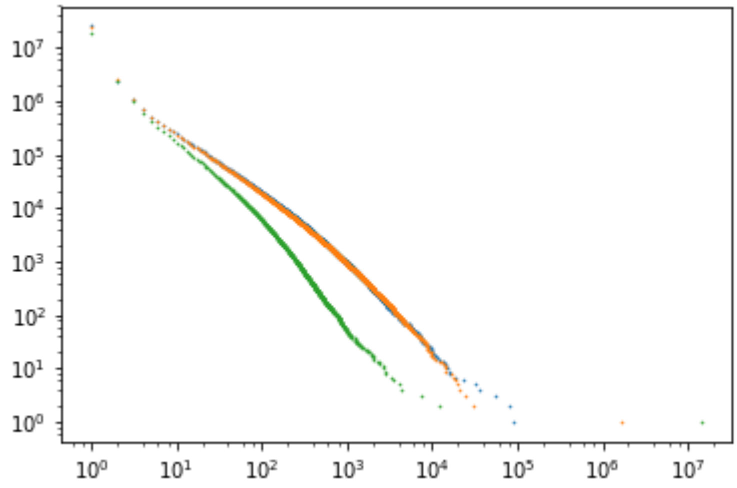
In [54]:
```
#Import des données
data={}
name=["ghtorrent","swh","swh_rootdir",]
for s in name:
    data[s]={}
    for ss in ["raw","cum"]:
        data[s][ss]={}
        with open(s+"_"+ss+".txt","r") as f:
            for line in f:
                x,y=line.split()
                data[s][ss][int(x)]=int(y)
        f.close()
#print(data)
```

In [55]:
```
# check global properties
no=0
for s in name:
    y_sum=0
    xy_sum=0
    for x,y in data[s]["raw"].items():
        y_sum+=y
        xy_sum+=x*y
    print(s,"# components",y_sum,"# origins",xy_sum)
    no=max(no,xy_sum)
print("ok if # of origins are =",no)
```

```
ghtorrent # components 25309069 # origins 41451739
swh # components 24017112 # origins 41451739
swh_rootdir # components 18536077 # origins 41451739
ok if # of origins are = 41451739
```

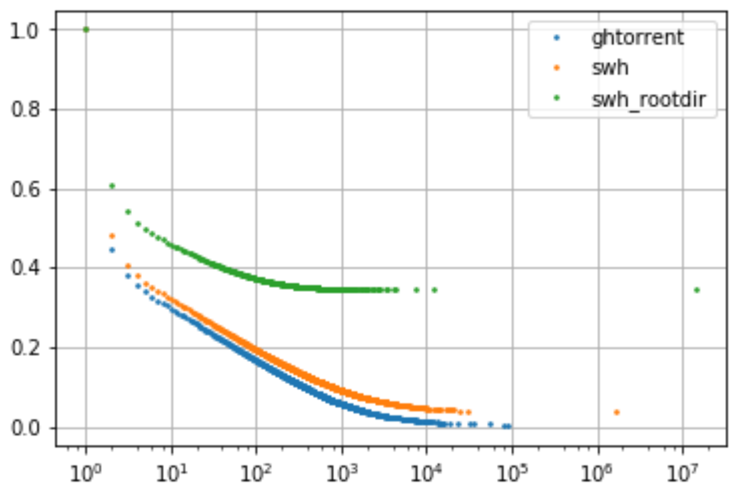
In [56]:
```
# affichage distribution cumulée des tailles des composants
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["cum"].keys()):
        x.append(key)
        y.append(data[s]["cum"][key])
    plt.loglog(x,y,"o",markersize=0.6)
    print(x[0],y[0])
plt.show()
```

```
1 25309069
1 24017112
1 18536077
```



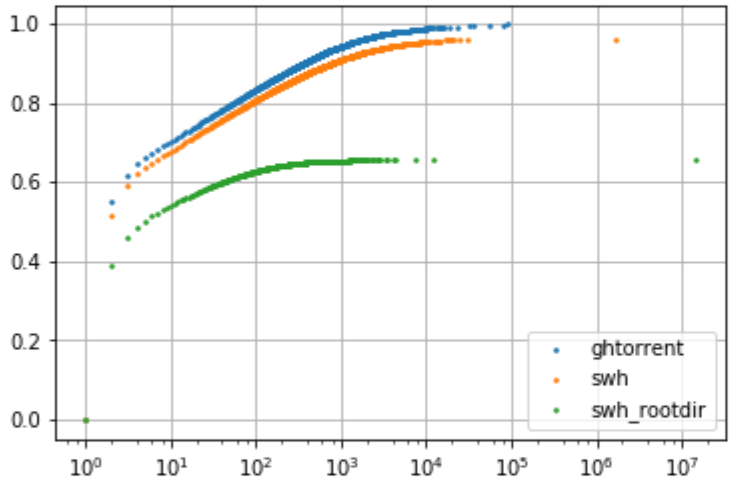
In [65]:
```
# affichage distribution cumulée des tailles des composants * la taille (ie le nombre d'origines dans des composants de tailles)
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum=np.zeros(len(y))
    # distribution cumulée de base
    for i in range(len(yysum)):
        yysum[i]=1*yy[i:].sum() # distribution non pondérée (identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
    plt.semilogx(xx,yysum/no,"o",markersize=1.6,label=s)
    print(xx[0],yysum[0])
plt.grid()
plt.legend()
plt.show()
```

```
1 41451739.0
1 41451739.0
1 41451739.0
```



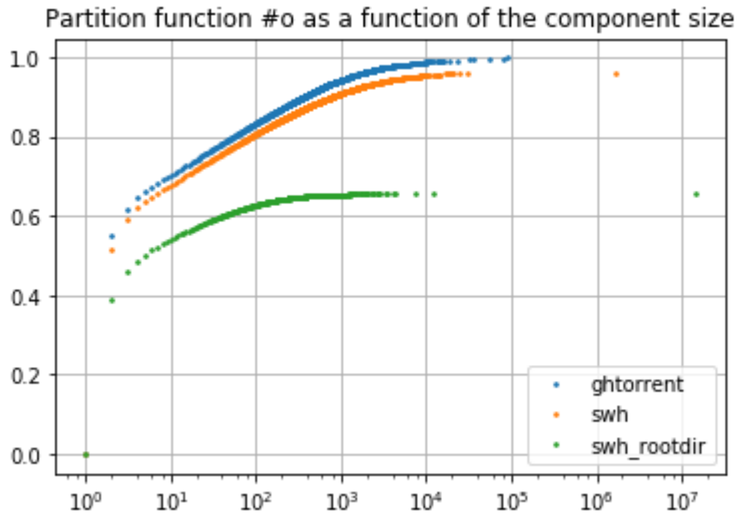
In [64]:
```
# affichage distribution cumulée des tailles des composants * la taille (ie le nombre d'origines dans des composants de tailles)
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum=np.zeros(len(y))
    # distribution cumulée de base
    for i in range(len(yysum)):
        yysum[i]=1*yy[i:].sum() # distribution non pondérée (identique à cum)
        yysum[i]=(x[i:]*yy[i:]).sum() # distribution pondérée
    plt.semilogx(xx,1-yysum/no,"o",markersize=1.6,label=s)
    print(xx[0],yysum[0])
plt.grid()
plt.legend()
plt.show()
```

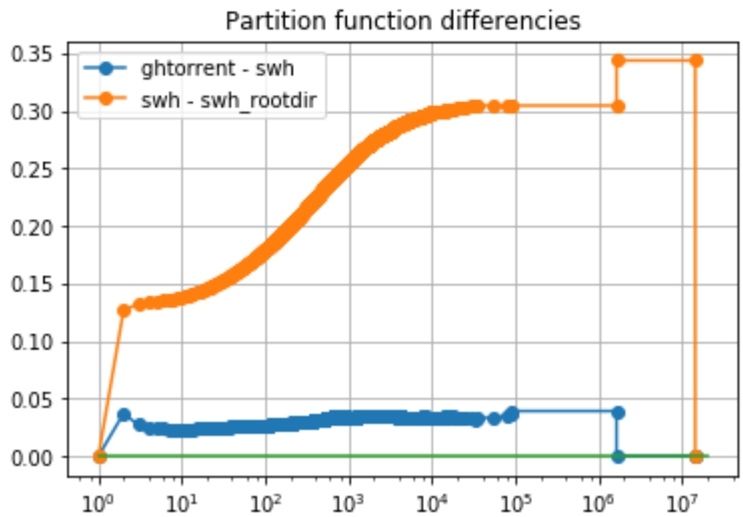
```
1 41451739.0
1 41451739.0
1 41451739.0
```

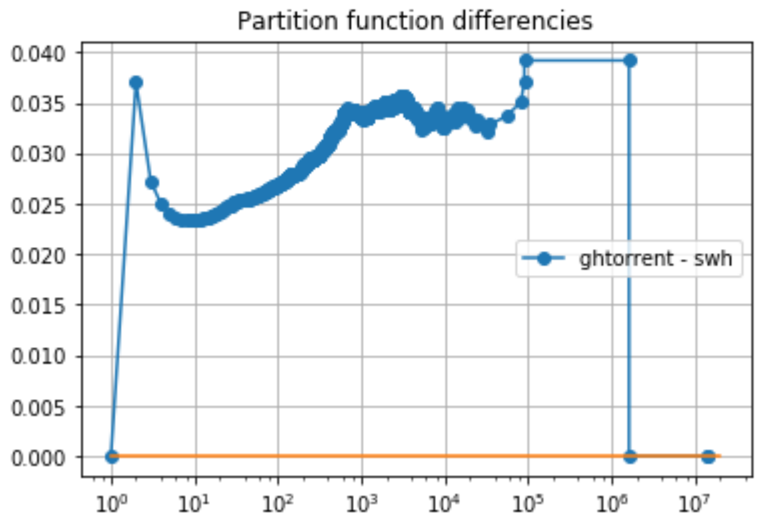


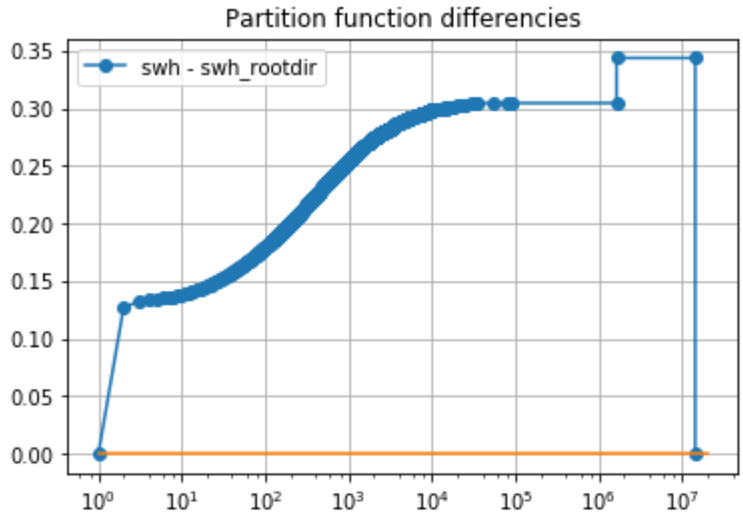
In [144]:
```
# on complete les histogrammes cumulées
#On crée une liste de toutes les valeurs de x
#puis on complete
# affichage distribution cumulée des tailles des composants *
la taille (ie le nombre d'origines dans des composants de tai
lles)
yysum={}
xxsum={}
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    xx=np.array(x)
    yy=np.array(y)
    yysum[s]=np.zeros(len(y))
    xxsum[s]=xx
    # distribution cumulée de base
    for i in range(len(yysum[s])):
        yysum[s][i]=1*yy[i:].sum() # distribution non pondéré
e (identique à cum)
        yysum[s][i]=1-(xx[i:]*yy[i:]).sum()/no # distribution
pondérée
    plt.semilogx(xxsum[s],yysum[s],"o",markersize=1.6,label=
s)
    print(xx[0],yysum[s][0])
plt.grid()
plt.legend()
plt.title("Partition function #o as a function of the compone
nt size")
plt.show()
# liste des x complete
xxsum2={}
yysum2={}
for s in name:
    for i in range(len(xxsum[s])):
        x=xxsum[s][i]
        y=yysum[s][i]
        try:
            xxsum2[x][s]=1
        except:
            xxsum2[x]={s:1}
        try:
            yysum2[x][s]=y
        except:
            yysum2[x]={s:y}
    try:
        xxsum2[x+1][s]=1
        yysum2[x+1][s]=1
    except:
        xxsum2[x+1]={s:1}
        yysum2[x+1]={s:1}

#print(yysum2)
#on bouche les trous
default={}
for s in name:
```

```
1 0.0
1 0.0
1 0.0
```


Partition function #o as a function of the component size
ghtorrent
swh
swh_rootdir


Partition function differencies
ghtorrent - swh
swh - swh_rootdir

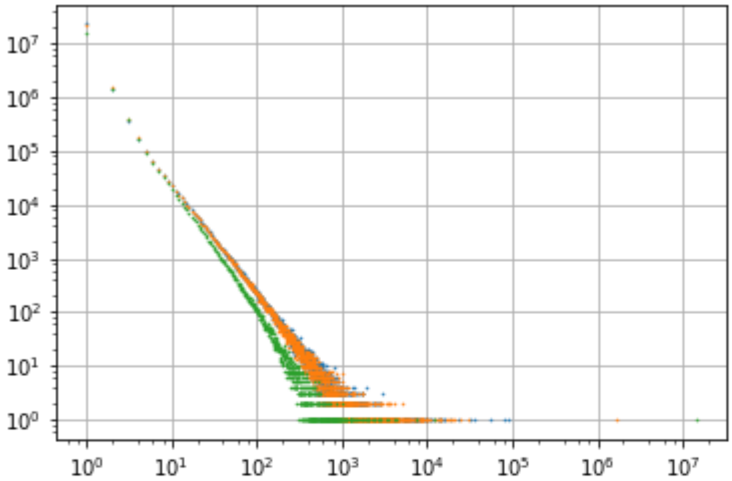

Partition function differencies
ghtorrent - swh


Partition function differencies
swh - swh_rootdir
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7

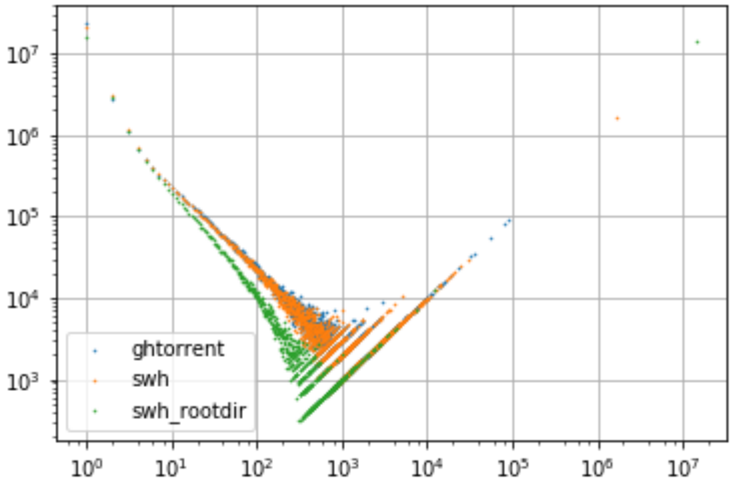
In [131]:

```
# affichage distributions BRUTES des tailles des composants
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key])
    plt.loglog(x,y,"o",markersize=0.6)
    print(x[0],y[0])
plt.grid()
plt.show()
for s in name:
    x=[]
    y=[]
    for key in sorted(data[s]["raw"].keys()):
        x.append(key)
        y.append(data[s]["raw"][key]*key)
    plt.loglog(x,y,"o",markersize=0.6,label=s)
    print(x[0],y[0])
plt.grid()
plt.legend()
plt.show()
```

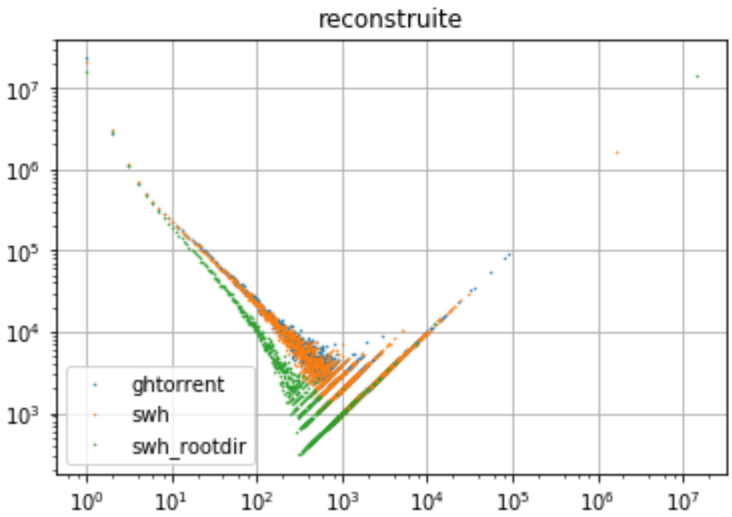
```
1 22906040
1 21372192
1 16131387
```




```
1 22906040
1 21372192
1 16131387
```



In [141]:
```
# distribution plot
for s in name:
    plt.loglog(xxsum3[0:len(xxsum3)-1],(yysum3[s][1:len(xxsum
3)]-yysum3[s][0:len(xxsum3)-1])*no,"o",markersize=0.5,label=
s)
plt.title("reconstruite")
plt.legend()
plt.grid()
plt.show()
```



In [136]:
```
print(xxsum3[len(xxsum3)-1])
print(xxsum3[len(xxsum3)-2])
```

```
14245628
14245627
```

In [139]:
```
for s in name:
    print(s,"-1",yysum3[s][len(xxsum3)-1])
    print(s,"-2",yysum3[s][len(xxsum3)-2])
```

```
ghtorrent -1 1.0
ghtorrent -2 1.0
swh -1 1.0
swh -2 1.0
swh_rootdir -1 1.0
swh_rootdir -2 0.656332222877
```

In [ ]: